NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

REAL-TIME SPEECH RECOGNITION SYSTEM FOR ROBOTIC CONTROL APPLICATIONS USING AN EAR-MICROPHONE

by

Dimitrios S. Koliousis

June 2007

Thesis Advisor:	Monique P. Fargues
Co-Advisors:	Ravi Vaidyanathan
	Peter Ateshian

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE		Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE June 2007	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE: Real-time Speech Recognition System for Robotic Control Applications Using an Ear-Microphone	5. FUNDING NUMBERS
6. AUTHOR(S) Dimitrios S. Koliousis	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited	12b. DISTRIBUTION CODE

13. ABSTRACT (maximum 200 words)

This study is part of an ongoing research started in 2004 at the Naval Postgraduate School (NPS) investigating the development of a human-machine interface command-and-control package for controlling robotic units in operational environments. An ear microphone is used to collect the voice-activated commands providing hands-free control instructions in noisy environments [Kurcan, 2006; Bulbuller, 2006].

This study presents the hardware implementation of a theoretical Isolated Word Recognition (IWR) system designed in an earlier study. The recognizer uses a short-term energy and zero-crossing based detection scheme, and a discrete Hidden Markov model recognizer designed to recognize seven isolated words. Mel frequency cepstrum coefficients (MFCC) are used for discriminating features in the recognizer phase. The hardware system implemented uses commercial off-the-shelf (COTS) electronic components, in-ear microphone, is portable and costs under $50.00.

The implemented speech capturing system uses the ear-microphone and the Si3000 Audio Codec to capture and sample speech clearly. The microprocessor processes the detected speech in real-time. The microprocessor's I/O devices work effectively with the audio codec and computer for sampling and training, without communication problems or data loss. The current implementation uses 1.181 msec to process each 15 msec data frame. Resulting recognition performances average around 73.72%.

14. SUBJECT TERMS Speech Recognition, Discrete Hidden Markov Model, Short-term Energy, Zero-crossing Measure, Real-time Isolated Word Recognizer, Robotic Control, Human-machine Interface, End-point Detection, Microprocessor, Microchip, Audio Codec, Speech Capturing System.	15. NUMBER OF PAGES 159
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

REAL-TIME SPEECH RECOGNITION SYSTEM FOR ROBOTIC CONTROL APPLICATIONS USING AN EAR-MICROPHONE

Dimitrios S. Koliousis
Captain, Hellenic Air Force
B.S., Hellenic Air Force Academy, Athens, 1992

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
June 2007

Author: Dimitrios S. Koliousis

Approved by: Monique P. Fargues
Thesis Advisor

Ravi Vaidyanathan
Co-Advisor

Peter R. Ateshian
Co-Advisor

Jeffrey B. Knorr
Chairman, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This study is part of an ongoing research started in 2004 at the Naval Postgraduate School (NPS) investigating the development of a human-machine interface command-and-control package for controlling robotic units in operational environments. An ear microphone is used to collect the voice-activated commands providing hands-free control instructions in noisy environments [Kurcan, 2006; Bulbuller, 2006].

This study presents the hardware implementation of a theoretical Isolated Word Recognition (IWR) system designed in an earlier study. The recognizer uses a short-term energy and zero-crossing based detection scheme, and a discrete Hidden Markov model recognizer designed to recognize seven isolated words. Mel frequency cepstrum coefficients (MFCC) are used for discriminating features in the recognizer phase. The hardware system implemented uses commercial off-the-shelf (COTS) electronic components, in-ear microphone, is portable and costs under $50.00.

The implemented speech capturing system uses the ear-microphone and the Si3000 Audio Codec to capture and sample speech clearly. The microprocessor processes the detected speech in real-time. The microprocessor's I/O devices work effectively with the audio codec and computer for sampling and training, without communication problems or data loss. The current implementation uses 1.181 msec to process each 15 msec data frame. Resulting recognition performances average around 73.72%.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

Initially I would like to appreciate Professors Monique P. Fargues, Ravi Vaidyanathan, and Peter P. Ateshian for their motivation and guidance during this thesis study.

Moreover, I would like to appreciate my friends Dr. Niko Janalato, Evangelo, Alejia, and Sofia Foutzitzi for their friendship.

In addition, I would like to appreciate my mother Vasiliki and my sister Loukia for their continuous support

This study could not be completed without my wife's Evanthia love and support. Thank you for your patience and understanding and for everything that you offer me these two years. The trip continues…

This thesis study is dedicated to my father Sotirios who was the first person in my life that believed to me and dreamed for me. Thank you father, Thank you for everything….

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

Automatic speech recognizers can be used to facilitate communication between humans and machines. Speech-based, human-machine interaction is demonstrated in several everyday applications, such as voice-mail systems in telephony, hands-free machine operations, communication interfaces for people with special abilities, dictation systems, and translation devices. ASR systems have been designed for different applications, in many areas, under a combination of restrictions, such as specific language and vocabulary, speaker dependency, noise-free environments and low talking rates, with excellent results.

This study is part of an ongoing research started in 2004 at the Naval Postgraduate School (NPS) which investigates the development of a human-machine interface command and control package applicable for soldier tele-operation of semi-autonomous military robots. The overall work is expected to be applied to an efficient, flexible, and robust human-machine interface system, capable of directing robotic units performing military missions, without debilitating, hampering, or interfering with a warfighter's field operations. An ear microphone was previously considered to provide control instruction corresponding to specific initiating actions in noisy environments [Kurcan, 2006; Bulbuller, 2006].

The objective of this study is the implementation of a real-time isolated word recognition system (IWR) using an ear-microphone as a human-machine interface for robotic control applications. Additional design specifications include the use of commercial off-the shelf (COTS) electronic components, portable small size and low cost.

This study followed three main phases; the theoretical model of an IWR speech recognition system, the hardware design and implementation of a real-time IWR system using an ear-microphone, and the system's software analysis, design and implementation.

The theoretical model of the IWR speech recognition system described here followed the study by [Kurcan, 2006]. This earlier work developed a 7-word IWR system

based on a discrete Hidden Markov model (DHMM) implementation. Discriminating features used to recognize the words are the mel frequency cepstrum coefficients (MFCC) and were selected as they have been shown to be robust to distortions [Kurcan, 2006]. The spoken words are cropped using end-points detection algorithms based on the short-tem energy (STE) and the zero-crossing measure (ZCR).

The dsPICFJ128GP256 Microchip's microprocessor is used in this study as the speech processing and recognition hardware solution. The selected ear microphone is the commercially available Ear Bone Microphone - XEM98D from IXCESSORY [IXCESSORY, 2007]. Finally our system's hardware uses the Si3000 audio codec from Silicon Laboratories for speech capturing which includes a pre-amplifier, amplifier, filter, and 15bit A/D converter contained in a compact case.

Software analysis, design and implementation are the basic steps required to transition the IWR theoretical processing model into a real-time microprocessor-based system. This study uses a combination of three different programming application platforms; MATLAB for training the DHMM model and testing the output of the microprocessor in different processing stages, DELPHI V5 for implementing the computer-microprocessor communication and for hexadecimal conversion of parameters resulted by MATLAB and hard-coded to microprocessor's program memory, and finally assembly for the microprocessor programming.

The study begun by examining the nature of speech signal and the methods used by the theoretical model for speech processing including the short-term Energy, the zero-crossing measure, and the mel frequency cepstrum coefficients (MFCC). In addition, the DHMM classification method was explored. Furthermore, the transformation of these methods in real-time algorithms was investigated to determine the required hardware resources (data and program memory, microprocessor clock speed, and I/O ports) and the data flow synchronization and timing issues.

In the real-time IWR system as implemented in this study, speech is constantly captured by the ear-microphone, amplified by 20 dB, passed through a low-pass filter (0 to 3500 KHz), sampled with 8 KHz sampling frequency, and transformed in 16-bit digital

samples by the speech capturing system. The binary samples are transmitted in packets of four samples to the microprocessor via the Data Converter Interface communication port activating the DCI interrupt which is used as the first theoretical timer of the real-time IWR system. Every time the DCI interrupt is triggered the program stores the samples in the word's buffer, update the word buffer pointer, and count the captured signal's samples. Next, the spoken word's start-point is detected. The theoretical model requires the ends-point detection routines to be executed in 80-sample speech frames overlapped by 50% or 40 samples. As a result, a second theoretical timer was introduced triggered by the DCI interrupt every 40 captured samples. Hence, every 40 samples the start-point detection routine is activating seeking for the beginning of the spoken word based on the STE and ZCR quantities. When a start-point is detected, the end-point detection, the feature extraction, the quantization, and the recognition routines are activated and ready to be executed in different times. The end-point detection algorithm is executed every 40 samples seeking for the end of capturing speech signal based on the STE and ZCR quantities. The feature extraction, quantization and recognition algorithms are executed in 256-sample overlap frames. The amount of overlap was chosen to be 53% or 136 samples. Consequently, a new 256-sample frame is formed every 120 new samples have been captured. Thus, every 120 samples or every three 40-sample timer triggers and while an end-point is not detected, the feature extraction, the quantization, and the recognition routines are executed. All these routines (including the end-point detection one) must be completed before the next activation of the 40-sample timer to preserve the real-time nature of the system. Finally when an end-point is detected, the recognition result is computed and transferred as binary command to the robotic control routine. More than 80% of processing time is available for control and command tasks before the system begin to process the new spoken word.

Three different approaches were used to evaluate the performance of the real-time IWR system; the off-line method, the real-time speaker-dependent method, and the real-time speaker-independent method. The overall recognition rate is 73.72%.

Furthermore, this study can be considered as the first step applying the previous research's results to a real-time system. Results show that the implementation of such a system can be a reality, opening the path for more research in this area.

I. INTRODUCTION

The objective of this study is the implementation of a real-time isolated word recognition system (IWR) using an ear-microphone as a human-machine interface for robotic control applications. Additional design specifications include the use of commercial off-the shelf (COTS) electronic components, portable small size and low cost.

This study is part of an ongoing research started in 2004 at the Naval Postgraduate School (NPS) which investigates the development of a human-machine interface command and control package applicable for soldier tele-operation of semi-autonomous military robots. The overall work is expected to be applied to an efficient, flexible, and robust human-machine interface system, capable of directing robotic units performing military missions, without debilitating, hampering, or interfering with a warfighter's field operations. An ear microphone was previously considered to provide control instruction corresponding to specific initiating actions in noisy environments [Kurcan, 2006; Bulbuller, 2006].

Our study can be considered as the first step applying these results to a real-time system. Results show that the implementation of such a system can be a reality, opening the path for more research in this area.

A. MAIN RESEARCH PHASES

This study followed three main phases; the theoretical model of an IWR speech recognition system, the hardware design and implementation of a real-time IWR system using an ear-microphone, and the system's software analysis, design and implementation.

The theoretical model of the IWR speech recognition system described here followed the study by [Kurcan, 2006]. This earlier work developed an IWR system based on a discrete-symbol Hidden Markov model implementation. Discriminating features used to recognize the words are the mel frequency cepstrum coefficients (MFCC) and were selected as they have been shown to be robust to distortions [Kurcan, 2006]. The spoken words are cropped using end-points detection algorithms based on the short-tem energy (STE) and the zero-crossing measure (ZCR). The overall recognition rate obtained

with Kurcan's off-line implementation was 92.7% [Kurcan, 2006], which motivated us to use the same type of approach for the real-time IWR system implementation considered in this study.

The dsPICFJ128GP256 Microchip's microprocessor is used in this study as the speech processing and recognition hardware solution. The microprocessor combines a digital signal processing (DSP) core (in addition to its central processor unit (CPU)), 32 Kbytes memory capacity, a large 256 Kbytes program memory, and different choices of I/O ports in the same small, low cost and easy to implement packet. It runs up to 40 million instruction cycles per second. In addition, it can be programmed using a variety of commands in assembly or in C language. The selected ear microphone is the commercially available Ear Bone Microphone - XEM98D from IXCESSORY [IXCESSORY, 2007]. Finally our system's hardware uses the Si3000 audio codec from Silicon Laboratories for speech capturing and includes a pre-amplifier, amplifier, filter, and 15bit A/D converter contained in a compact case.

Software analysis, design and implementation are the basic steps required to transition the IWR theoretical processing model into a real-time microprocessor-based system. In real-time systems implementations analysis is the most critical step, because it is the step where all the theoretical and technical information of an Automatic Speech Recognition (ASR) system are combined to produce proper real-time processing algorithms. The result of this step is a complete analysis block diagram which describes the signal flow from the point of capturing a signal of interest to the delivery of final commands to the machine. Design is the step where every "black box" of the analysis block diagram is divided into its components, and software and hardware decisions are made for their implementation, such as programming language, memory buffer size, DSP core usage and I/O port requirements. Finally, in the implementation step, the design components are transformed into microprocessor programs and are loaded into memory. This step also incorporates all required tests for system evaluation. This study uses a combination of three different programming application platforms; MATLAB for training the discrete-symbol hidden Markov model (DHMM) and testing the output of the microprocessor in different processing stages, DELPHI V5 for implementing the computer-microprocessor communication an for hexadecimal conversion of parameters

resulted by MATLAB and hard-coded to microprocessor's program memory, and finally assembly for the microprocessor programming.

B. THESIS ORGANIZATION

Chapter I introduces the main objective of this research related to previous studies conducted in this area. It also briefly describes the main phases followed during the development of the real-time IWR system.

Chapter II presents the nature of speech signal and the methods commonly used nowadays for speech processing including the short-term Energy, the zero-crossing measure the linear prediction coefficients (LPC), the real cepstrum analysis (RC), and the mel frequency cepstrum coefficients (MFCC). Moreover, it provides an introduction to the automatic speech recognition (ASR) problem and briefly describes the hidden Markov Model approach used in ASR systems (training and recognition). The chapter ends by describing the main hardware and software challenges faced in the development of our real-time IWR system.

Chapter III describes the hardware used in this study for the real-time implementation of the IWR system including the speech capturing system and the speech processing unit.

Chapter IV presents the software real-time implementation of the IWR system describing in details the main program, responsible for the initialization and synchronization of the hardware to work as a real-time speech processing and recognizer. Furthermore, a simple solution for robotic control is proposed. Finally, the error identification routines are explained in details.

Chapter V describes the real-time end-point detection routines as implemented in assembly language in this study.

Chapter VI represents the real-time implementation of the speech feature extraction algorithms. Two different routines are used to obtain the MFCC parameters of the captured speech; the feature extraction activation procedure which is responsible for the synchronization in a frame base of the input signal with the processing routines, and the feature extraction procedure which consists of the actual assembly routines used for the MFCC parameters extraction.

Chapter VII describes the speech features vector quantization procedure. This step has two parts; first a MATLAB implementation to calculate the quantization codebook, and second the assembly implementation of the features vector quantization in the microprocessor.

Chapter VIII presents the real-time isolated word recognition part of our system. First, the theoretical recognition model based on the DHMM approach is analyzed followed by the model's training algorithm as implemented in MATLAB. Finally, we describe the assembly implementation of the real-time isolated word recognizer.

Chapter IX includes the real-time IWR system operation and evaluation. In this chapter, the main operation instructions are given for the system's user. In addition, the different evaluation methods are analyzed. Finally, the evaluation results for the system performance are presented.

Finally, Chapter X presents conclusions and recommendations for future research.

II. SPEECH PROCESSING AND RECOGNITION

Automatic speech recognition (ASR) systems consist of two major parts: the speech processing and the recognition. In this Section we briefly describe the main issues associated with these two ASR's components.

Speech is a non-stationary signal and processing and is usually conducted over short-time frames where stationarity can be assumed. For example, linear prediction approaches model the speech by a set of coefficients which represent the filter coefficients of an all-zero model of the human speech production system. Cepstrum analysis, on the other hand, uses homomorphic transformations to extract speech features and apply well known spectrum linear operations. "Psychoacoustic" properties were taken into account along with cepstrum analysis to derive the Mel Frequency Cepstral Coefficients (MFCC), which are widely used in speech recognition nowadays.

This section introduces the basic idea behind Automatic Speech Recognition (ASR) in the context of isolated word and continuous speech recognition applications. The ASR implementation considered in this study also uses vector quantization to code speech features to reduce possible redundancies and is also one of the steps needed in the derivation of Discrete Hidden Markov Models (HMMs) commonly used nowadays in ASR. HMM is a "stochastic approach" which is used in this study to model the different words selected here. Finally, we discuss real-time, isolated-word recognizer implementation applications.

A. FUNDAMENTALS OF SPEECH

1. Speech Definitions

Speech is a basic way of communication between humans. As a human action, speech communication can be characterized as a sequence of the following events:

- Speaker's brain produces a "thought",
- Brain expresses the thought in a series of words according to specific *linguistic rules*,
- Brain produces a series of commands to activate the *speech production mechanism* in a specific way to produce sounds that represent these words,

- Speech production mechanism organs move according to the brain commands and produce the desired sound pressure waves,
- Sound waves travel in air,
- Sound waves are captured by the listener's auditory system and translated into neurological signals,
- Listener's brain translates the neurological signals into words according to the same linguistic rules used by the speaker to form the speech,
- Listener brain uses these words to form a "thought." This "thought" is likely to be the same as the initial speaker's.

From an engineer's perspective, a "speech waveform is an acoustic sound pressure wave that originates from voluntary movements of anatomical structures" [Deller, 1993]. Speech is a time-varying, and consequently, non-stationary signal. In addition, speech characteristics are also affected by other factors such as the speaker's anatomy, age and current physical situation. To mitigate these problems, speech signal processing can be extended to include the study of the human speech production mechanism. Moreover, investigation of the human auditory system is necessary to understand the way humans perceive and process speech signals. A general description of the speech production mechanism and the main characteristics of speech follow.

2. Human Speech Production System

Figure 1 shows the anatomy of the human speech production mechanism. Two main regions can be distinguished in this figure, the subglottal system and the supra-laryngeal vocal tract.

- The *subglottal system* consists of:
  - The diaphragm,
  - The lungs,
  - The trachea,
  - The esophagus,
  - The larynx.
- The *supra-laryngeal vocal tract* consists of three different cavities:
  - Pharyngeal cavity,

- Oral cavity,
- Nasal cavity.

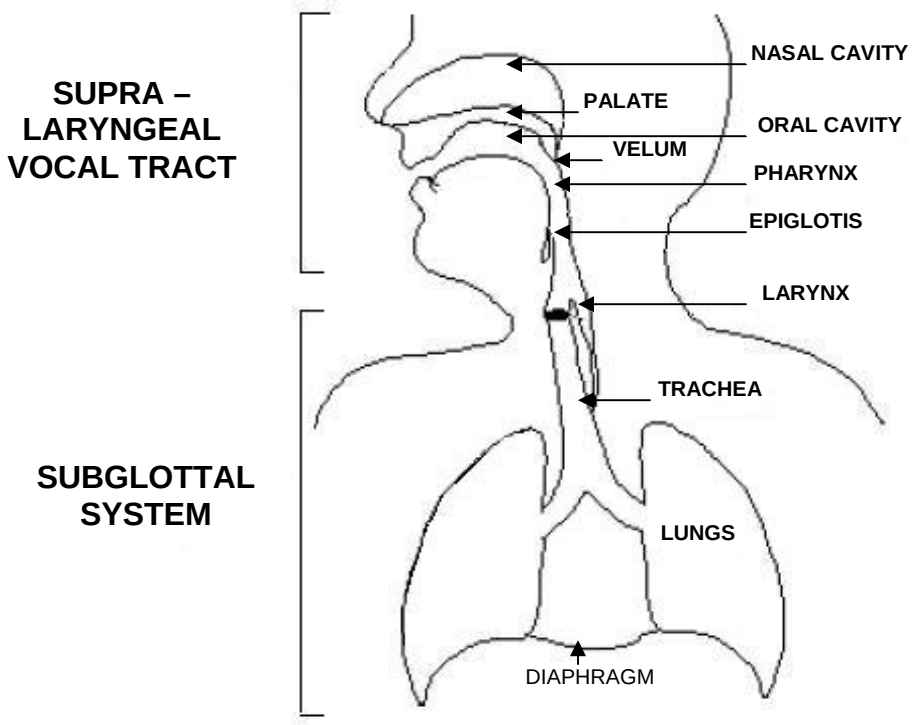

Figure 1. Anatomy of human speech production mechanism [After Tummala, 2007]

The subglottal system can be thought of as the speech excitation source which can be viewed as the input to a filter. The filter's characteristic transfer function is affected by the three cavities and their components such as tongue, teeth, lips and velum. These components are called *articulators*. Their movements are responsible for the resulting shape of sound waves radiating from the speaker.

3. General Speech Characteristics

Voiced and *unvoiced* speech result from two different types of excitation. Voiced sounds are produced when air from the lungs passes through the larynx. The larynx oscillates the air in a specific way. Next, the air passes through the vocal tract area resulting in a speech sound known as *voice* or *phonation*. The *fundamental period* and *frequency* characterize the phonation and are related to the duration and the rate of vocal fold openings, respectively. The fundamental frequency is also called the *pitch*, and

differs among people due to the differences in their larynx's anatomy. Men's pitch range occurs commonly in the interval between 50 to 250 Hz while women's lies between 120 and 500 Hz [Deller, 1993].

Unvoiced speech sounds are produced by air passed directly through vocal tract formations. Unvoiced speech, contrary to voiced speech, does not exhibit periodicity, and is characterized by a noise-like signal. Usually, unvoiced speech sounds have higher frequencies and less energy than voiced ones [Fargues, 2005].

Phonemes represent the linguistic units of spoken language. Phonemes are combined to transform words, which describe a "thought," into speech. Every language consists of a specific set of phonemes. The English language, for instance, consists of 42 different ones. Each phoneme declares a specific combination of *articulator gestures* also including the excitation manner. Based on the articulator gestures and the excitation, different groups of phonemes exist such as fricative, vowel, stop, liquid, diphthong, glide and nasal. The actual sounds of phonemes are called *phones* [Deller, 1993].

Phonemes are discrete units and their actual sound (phones) can be easily recognized when they are spelled independently. On the contrary, this task becomes much more difficult in continuous speech scenarios, as the speech producing organs cannot immediately transition from one gesture to another. This phenomenon is called *coarticulation* and it is one of the parameters which affect ASR performance. In addition, the speech signal bandwidth is mostly limited to 4-5 KHz due to restrictions of the articulators' anatomy.

The vocabulary used in this thesis study consists of the words *"left," "right," "up," "down," "move," "pan,"* and *"kill."* The phonemes and phonetic properties of these words are represented in Table 1.1.

Word	Phoneme	Manner of articulation	Voiced?
Left	l	Liquid	Yes
	ϵ (eh)	Vowel	Yes
	f	Fricative	No
	t	Stop	No
Right	r	Liquid	Yes
	αι (ay)	diphthong	Yes
	T	Stop	No
Up	Λ (ah)	Vowel	Yes
	P	Stop	No
Down	d	Stop	Yes
	αu (aw)	diphthong	Yes
	W	Glide	Yes
	N	Nasal	Yes
Move	M	Nasal	Yes
	u (uw)	Vowel	Yes
	V	Fricative	Yes
Pan	P	Stop	No
	æ (ae)	Vowel	Yes
	N	Nasal	Yes
Kill	K	Stop	No
	i (iy)	Vowel	Yes
	L	Liquid	Yes

Table 1.1 English Phonemes and Characteristics for Vocabulary Words of Interest [After Kurcan, 2006]

In the next section, short-term speech digital signal processing methods are analyzed including the Linear Prediction (LP) and Real Cepstrum (RC) analysis.

B. SPEECH PROCESSING

1. Short-term Speech Digital Signal Processing

As mentioned earlier, speech is a non-stationary signal. To overcome this problem, speech is partitioned into small time intervals where stationarity may be assumed. The speech intervals are called *frames* while the partitioning procedure is named *short-term processing*. The idea behind short-term processing is "to analyze the

frames of speech as those frames move through time and attempt to capture transient features of the signal" [Deller, 1993]. Speech frames are produced by windowing. The type of window and its length are important as they affect processing results. In this thesis study, a hamming window of length 256 speech samples (or 32 msec of speech at 8 KHz sampling frequency) with 53% overlap between consecutive frames is employed.

This thesis also uses short-term digital signal processing for end-point detection and for speech feature extraction. Specifically, the short-term methods used for the ASR implementation, includes short-term signal energy measure (STE), zero crossing measure (ZCM), Fast Fourier Transform (FFT), and discrete cosine transform (DCT). More information about these procedures is given in Chapters V and VI.

2. Speech Linear Prediction Analysis

Speech linear prediction analysis (LP) is a digital signal processing technique for speech analysis. Currently, it is mainly used in speech compression schemes but was also widely used for automatic speech recognition in the past. Using knowledge of the human speech production system and its output (speech), mathematical models are developed to describe the speech signal. Linear prediction speech analysis is based on the inversion of the speech synthesis problem. The method is represented here for completeness and as an introduction to speech processing.

Speech synthesis occurs in a human speech production system and can be described by the "terminal-analog model" [Deller, 1993] shown in Figure 2. The term "terminal-analog" means that the model and the system that it represents are analogous only at their outputs' product (speech waveforms), and the intermediate stages cannot be considered analogous. In this model, the speech synthesis system is a filter with different types of excitation as input and speech waveforms as output. The system transfer function $H(z)$ for each speech frame (where stationarity can be considered) is given in Equation (2.2.1) as

$$H(z) = H_0 \frac{1 + \sum_{i=1}^{L} b(i) z^{-i}}{1 - \sum_{i=1}^{R} a(i) z^{-i}}. \qquad (2.2.1)$$

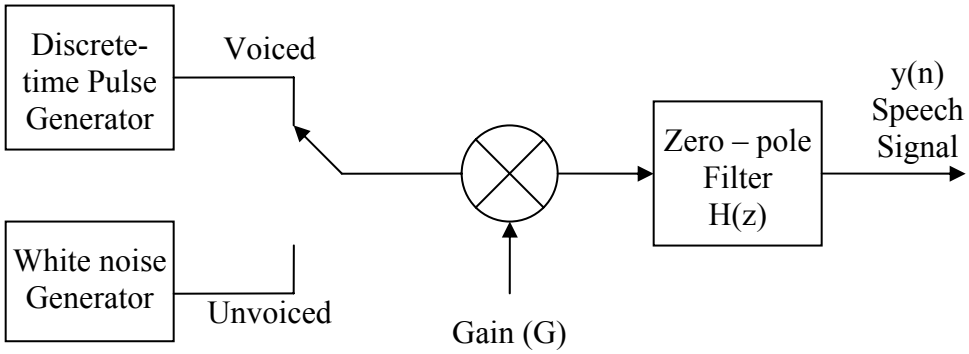

Figure 2. Discrete-time "terminal-analogous" Speech Production Model

The input of this system (the excitation source) is the sequence $v(n)$ given in equation (2.2.2) below

$$v(n) = \begin{cases} \sum_{k=-\infty}^{\infty} \delta(n - kP), & \text{voiced case} \\ \text{white noise Normal(0,1)}, & \text{unvoiced cases,} \end{cases} \qquad (2.2.2)$$

where *P* is the order of the filter and *n* is the signal index.

In LP synthesis, the zero-pole transfer function *H(z)* is replaced by an all-pole transfer function, with the same spectrum magnitude, but not necessarily the same spectrum phase. Defining the synthesis filter as a pure AutoRegressive (AR) filter allows to define the speech analysis filter as simple FIR filter. In addition, the resulting synthesis filter can be made stable by moving all poles inside the unit circle. This last result is based on the fact that any zero outside the unit circle can be replaced by a corresponding reciprocal conjugate zero located inside the unit circle. Next, all zeros are cancelled by adding a pole in their position. Finally, the resulting transfer function of order *M* has *M-poles* inside the unit circle and *M-zeros* at the origin. The transfer function $\hat{H}(z)$ of the

analysis model shown in Figure 3 is given by Equation (2.2.3), where we note that the filter coefficients $\hat{a}(i)$ in (2.2.3) are different from $a(i)$ in Equation (2.2.1).

$$\hat{H}(z) = \hat{H}_0 \frac{1}{1 - \sum_{i=1}^{M} \hat{a}(i) z^{-i}}. \tag{2.2.3}$$

The all-pole speech synthesis estimation model is valid based on the fact that the human auditory system is "phase deaf" [Milner, 1970]. Note that this does not mean that humans cannot detect phase differences in speech signals, as the phase difference information is used to estimate the speech source locations. In this context "phase deaf" means that speech perception is mainly dependent on magnitude quantities. For instance, humans who are deaf in one ear are able to perceive speech, but are not capable of determining the location of the speech source.

The all-pole transfer function $\hat{H}(z)$ defined in Equation (2.2.3) is an approximation of the zero-pole transfer function $H(z)$ shown in Equation (2.2.1). The filter coefficients $\hat{a}(i)$ are computed as follows. Assume that $y[n]$ is a wide-sense stationary random process. The linear prediction problem is defined as the estimation of the n^{th} value of y ($y[n]$) using the previous P values of y ($y[n\text{-}1]$, $y[n\text{-}2]$, …, $y[n\text{-}P]$), where P is the *order* of the system [Therrien, 2004]. It is expressed by

$$\begin{aligned} \hat{y}(n) &= -\hat{a}^*(1) y(n-1) - \hat{a}^*(2) y(n-2) + \ldots - \hat{a}^*(P) y(n-P) \\ \hat{y}(n) &= -\sum_{i=1}^{P} \hat{a}^*(i) y(n-i). \end{aligned} \tag{2.2.4}$$

Evaluating the performance of the linear predictor can be measured by the *error* $e(n)$ defined as the difference between the actual value $y[n]$ and its predicted value $\hat{y}[n]$ and is written as

$$e(n) = y(n) - \hat{y}(n), \tag{2.2.5}$$

or

$$e(n) = y(n) + \hat{a}^*(1) y(n-1) + \hat{a}^*(2) y(n-2) + \ldots + \hat{a}^*(P) y(n-P). \tag{2.2.6}$$

The parameters $\hat{a}(i)$ are obtained by minimizing the average squared error which leads to solving the *Normal Equations* defined as:

$$\tilde{R}_y \hat{a} = b, \tag{2.2.7}$$

where

$$\tilde{R}_y = E\left\{\tilde{y}\tilde{y}^{*T}\right\}\begin{bmatrix} R_y[0] & R_y[1] & \cdots & R_y[P] \\ R_y[-1] & R_y[0] & \cdots & R_y[P-1] \\ \vdots & \vdots & \ddots & \vdots \\ R_y[-P] & R_y[-P+1] & \cdots & R_y[0] \end{bmatrix},\ \hat{a} = \begin{bmatrix} 1 \\ \hat{a}(1) \\ \vdots \\ \hat{a}(P) \end{bmatrix} \text{ and } b = \begin{bmatrix} \sigma_e^2 \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \tag{2.2.8}$$

Several methods may be applied to solve the normal equations. The basic method to calculate the inverse of the matrix $\tilde{R}$ is computationally expensive, especially for higher order filters. Thus, recursive methods have been developed to reduce the computational effort such as the Levinson–Durbin Recursion method and others [Therrien, 2004].

Linear prediction coefficients are also widely used in speech processing to estimate characteristics such as the pitch and the formants of a speech signal. For example, the Simple Inverse Filter Tracking (SIFT) is a pitch estimation method [Markel, 1972] which uses linear prediction to pre-whiten speech frames. It is also used to solve the voiced–unvoiced characterization of speech. Other simpler techniques apply the Fast Fourier Transform to the LP coefficients and use the spectrum magnitude peaks to estimate formants [Hanauer, 1971], [Markel, 1972], [Kang and Coulter, 1976], etc…

Two main issues are associated with LP methods used for speech parameters estimation. First is determining the filter order *P* and, second selecting the actual speech signal excitation. LP analysis was developed as a method to estimate the parameters of the all-pole filter *H*(*z*). The order of this filter is unknown, as it describes a terminal analogue model where only the output signal speech is given. Theoretically, the LP model works ideally as *P* goes to infinity and/or as pitch frequency goes to zero. In real situations, LP analysis demonstrates good results for low-pitch, voiced speech such as "male" phonation, but exhibits poorer performance for high-pitch, voiced speech, i.e., female or child phonation. Different techniques have been developed for the choice of

order *P* such as the Markel and Gray method expressed in Equation (2.2.9) [Markel and Gray, 1976]. Finally, Figure 3 shows a block diagram of the LP analysis method. The resulting filter coefficients produced by error minimization are used to characterize the desired speech.

$$P = \begin{cases} floor\left[F_s + (4 \text{ or } 5)\right], & \text{voiced} \\ floor\left[F\right]_s, & \text{unvoiced} \end{cases} \tag{2.2.9}$$

where F_s = Sampling frequency in KHz.

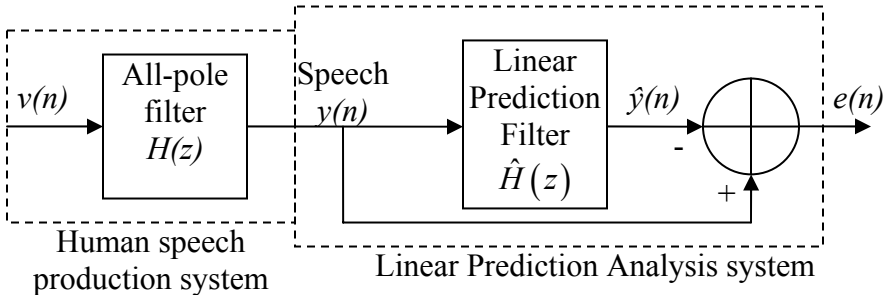

Figure 3. Linear Prediction Analysis block diagram

3. Real Cepstrum Analysis

Real Cepstrum (RC) analysis is used in this thesis as a short-term processing method for speech feature extraction. Basically, cepstrum analysis transforms signals in such a way that non-linearly combined signals may be separated. Speech, as modeled in Section II.B.2 and as shown in Figure 2, is expressed as the convolution of the excitation signal *v*(*n*) with the vocal tract's filter transfer function *h*(*n*):

$$y(n) = v(n) * h(n). \tag{2.2.10}$$

Classic spectrum analysis cannot deal with problems such as the separation of $v(n)$ and $h(n)$, as the convolution is a non-linear operation and $v(n)$ and $h(n)$ are unknown. Cepstrum analysis uses a "homomorphic" transformation to form a linear relationship between these two signals in such a way that the resulting signals become separable.

There are two types of Cepstrum transformations: *Real Cepstrum* (RC) and *Complex Cepstrum* (CC). Their main difference is in the transformation operation they use. The RC uses the logarithm of the component signals' spectrum magnitude while the CC uses the logarithm of the complex spectrum. The logarithm of a complex number (or complex logarithm) is given by the following equation:

$$\log(z) = \log(|z|) + j \arg\{z\}, \quad \text{where } z \text{ is a complex expression.} \tag{2.2.11}$$

As a result, CC can be considered a "homomorphic" transformation fulfilling the demand of reversibility, while the RC cannot, due to the loss of the phase information during the logarithm operation. However, the RC is easier to implement than the CC and widely used in cases where the recovery of the original non-linear signal is not critical, such as in speech recognition. This is the case in this study.

Real Cepstrum analysis can be described by three steps. First, the fast Fourier Transform (FFT) is calculated for each speech frame. As a result, the time-domain convolution shown in Equation (2.2.10) becomes a multiplication in the frequency domain as follows:

$$Y(z) = V(z) \cdot H(z). \tag{2.2.12}$$

Second, the logarithm of the right and left components of Equation (2.2.12) are calculated as follows:

$$\log|Y(\omega)| = \log|(V(\omega) \cdot H(\omega))| \Rightarrow \log|Y(\omega)| = \log|V(\omega)| + \log|H(\omega)|, \tag{2.2.13}$$

or

$$C_y(\omega) = C_v(\omega) + C_h(\omega). \tag{2.2.14}$$

At this point, the transformation has been completed and the speech components are linearly combined. Thus, in the third step, the well-known linear operations of spectrum analysis such as the inverse FFT are applied, resulting in the following equation:

$$c_y(n) = c_v(n) + c_h(n). \tag{2.2.15}$$

It is interesting to note the indices change from frequency (ω) in Equation (2.2.14) to time (n) in Equation (2.2.15). The components of Equation (2.2.15) are in what is referred to as the "quefrency" domain, to be distinguished from the time domain.

Many well-known signal processing techniques such as filtering or "liftering," as it is called in Cepstrum analysis, can be applied to the linearly-combined signal $c_y(n)$ in the "quefrency" domain. The idea is to apply a filter ("lifter" in the cepstrum vocabulary) to eliminate well-separated components of the signal. This procedure is presented in Figure 4, where a "lifter is applied to the cepstral coefficients $c_y(n)$ to obtain $c_h(n)$. Furthermore, the "quefrency" domain can be used for pitch [Noll, 1967] and formant estimation [Schafer & Rabiner, 1970], as shown in Figure 5, where the cepstrum peaks are used for pitch period estimation of the voiced phoneme "ae" in the word "pan." In this case, the difference between the peaks is 40 samples or 5 msec. In addition, Figure 6 shows that information is mainly contained in the first cepstrum coefficients, indicating that only a few of them may be sufficient to characterize the "i" phoneme of the word "kill."

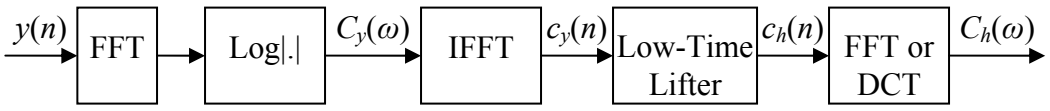

Figure 4. Low-time Liftering of $C_y(\omega)$ to Obtain the Component "Signal" $C_h(\omega)$.

Even though, the LP coefficients are able to describe the speech signal, as was described earlier, they also have limitations due to speaker- and excitation -type dependency. Real cepstrum analysis is preferred to extract speech parameters (features) which can be classified and applied to a speech recognizer. Thus, real cepstrum approaches are used to overcome LP analysis problems, and lead to new parameters, such as Mel-Cepstrum parameters, and Delta or Differenced Cepstrum parameters. This study uses the Mel-Cepstrum parameters as inputs to an ASR implementation because they were shown in an earlier study to lead to better recognition performances in the selected Discrete Hidden Markov Model recognizer than the other two types of features considered for the data investigated [Kurcan, 2006].

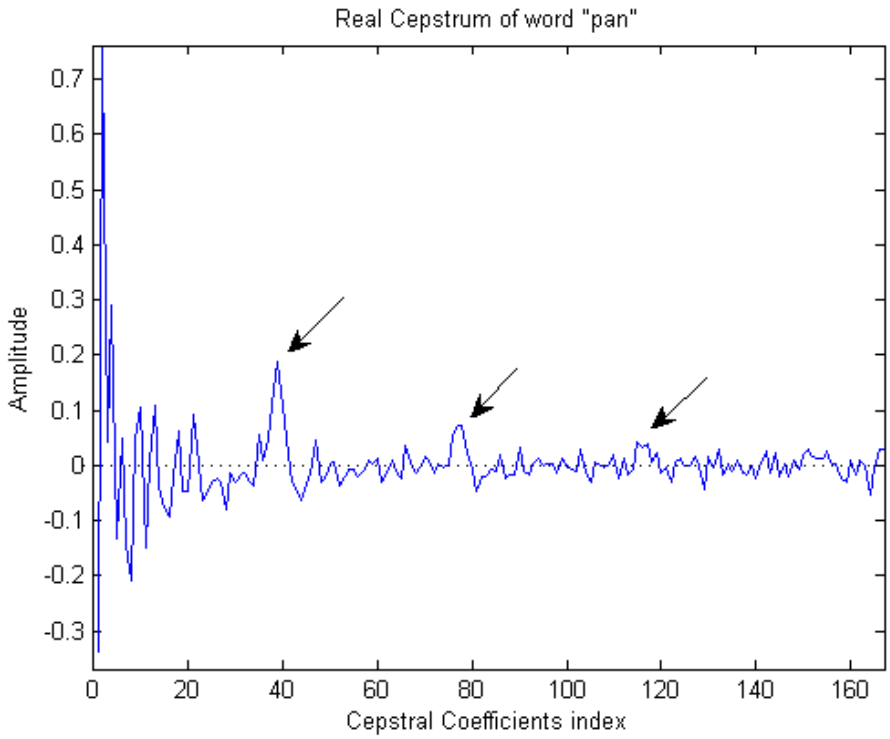

Figure 5. Real Cepstrum of word “pan”

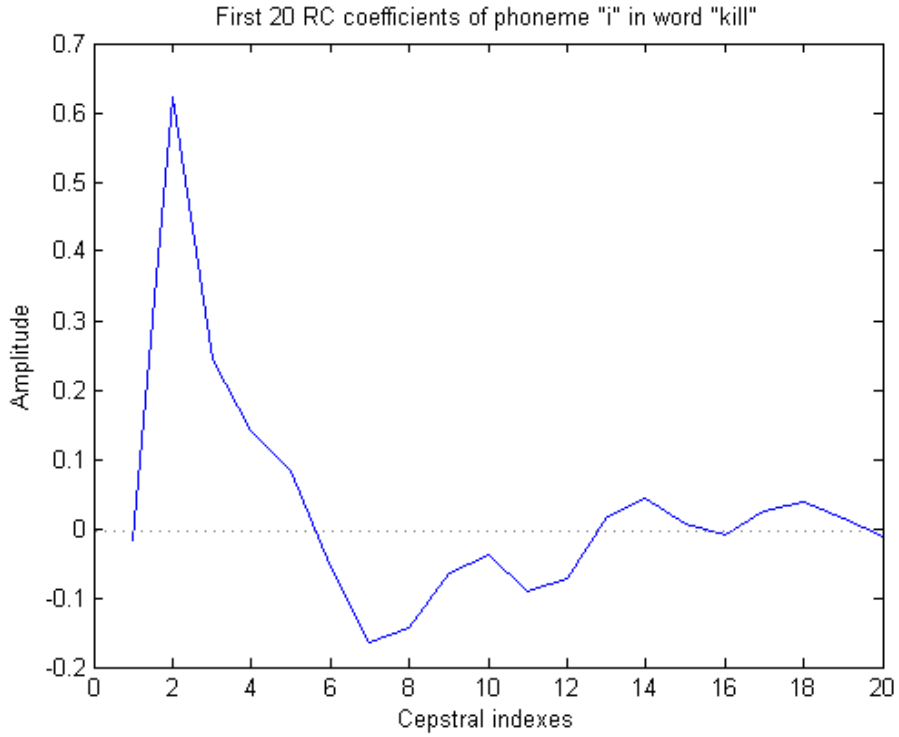

Figure 6. First twenty Real Cepstrum coefficients of phoneme “i” in word “kill.”

4. Mel-Cepstrum Analysis

The Mel-Cepstrum parameters calculation is based on "psychoacoustics," which takes into account how speech is perceived by the human auditory system. Research studies have shown that a human's ear does not linearly perceive audio frequencies and that a frequency of high magnitude can hide a neighboring frequency with lower magnitude. Mel Scale Mapping and Filter Bank approaches are results of these observations. The unit of the Mel Scale (called *mel)* mimics the human perceived frequency, and slightly different mapping equations have been proposed over the years to relate mel units and actual signal frequencies. The mapping equation used in this thesis [O'Shaughnessy, 1987; Picone, 1993] is given by

$$F_{Mel} = 2595 \cdot \log_{10}\left(1 + \frac{f_{actual}(Hz)}{700}\right). \qquad (2.2.16)$$

Figures 7 and 8 show a plot of the mel scale between 0 and 5KHz, and a zoomed in version between 0 and 1 KHz, respectively. Note that the mapping is fairly linear between 1 and 1000Hz and exhibits logarithmic behavior at higher frequencies.

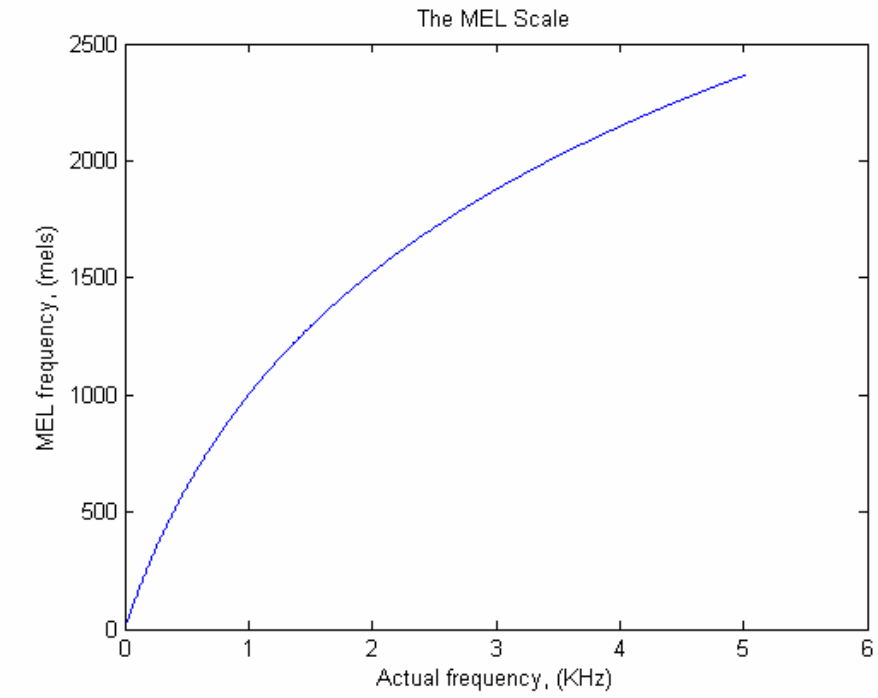

Figure 7. The mel Scale Corresponding to Equation 1.2.16.

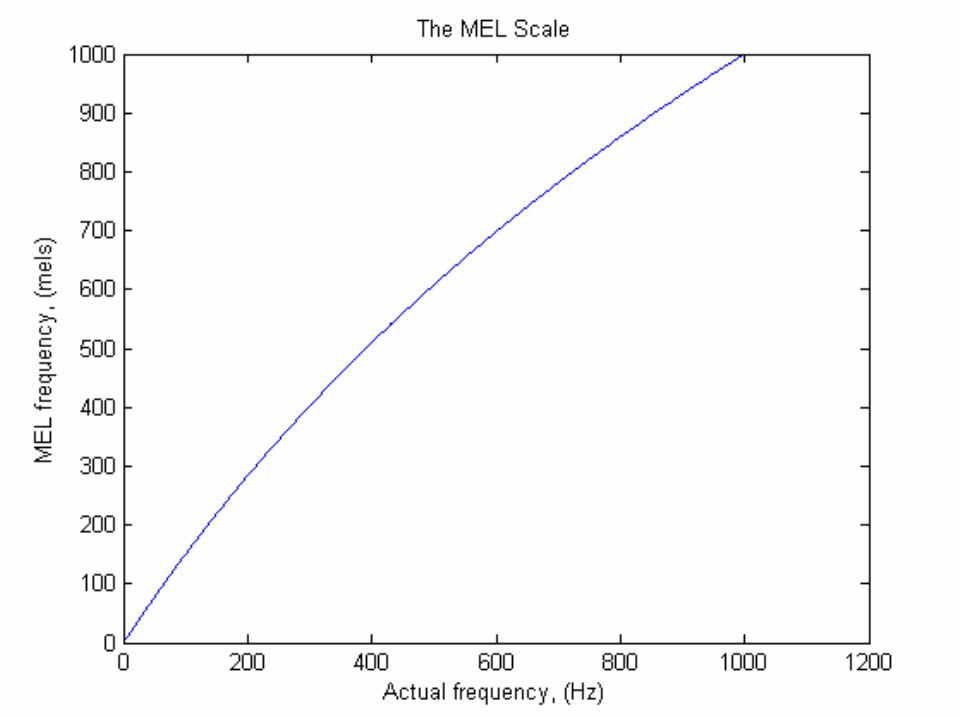

Figure 8. The Linear Part of the mel Scale from 1 to 1000 Hz

Critical band filter banks are based on the "psychoacoustics" experimental observation of the domination of neighboring frequencies [Picone, 1993]. According to this property, human speech perception can be described by a group of simple band-pass FIR filters with a specific bandwidth. Many types of filter banks schemes exist and differ mainly in the choice of central frequencies and filter type. The *Mel critical band filter bank used in this study* consists of a set of 24 overlapped triangular FIR band-pass filters with central frequencies derived from the mel scale, as proposed in [Davis, 1980]. The central frequencies and the corresponding bandwidth of this implementation are listed in Table 1.2. The frequency domain representation (FFT) of the triangular filters is shown in Figure 9.

The Mel Filters Cepstrum Coefficients (MFCC) are captured as follows:

- Apply the FFT to the speech frame,
- Filter the resultant speech bins in the frequency domain using the 24 mel filter banks,
- Calculate the weighted average magnitude of the resultant bins for every filter bank. This can be considered a calculation of the weighted average energy of the speech signal into the bandwidth which is defined by each filter bank,

- Take the logarithm of the energy of the pass-band signal,
- Calculate the MFCC parameters using the IFFT or the discrete cosine transform (DCT). Note that the IFFT can be replaced by the DCT as it is applied to symmetric and real variables [Deller, 1993], resulting in highly uncorrelated parameters [Jayant, 1984; Deng 2003].

This procedure is shown in Figure 10 and is mathematically described by:

$$MFCC_n = \sum_{k=1}^{20} X_k \cos\left[n\left(k - \frac{1}{2} \right) \frac{\pi}{L} \right], \quad \text{for } n = 0,1,2,...,M \quad \text{and} \quad k = 1,2,...L \qquad (2.2.17),$$

where M is the total number of MFCC parameters, L is the total number of mel filter banks and X_k is the logarithm of the weighted average of the speech energy in the bandwidth defined by the k^{th} mel filter bank.

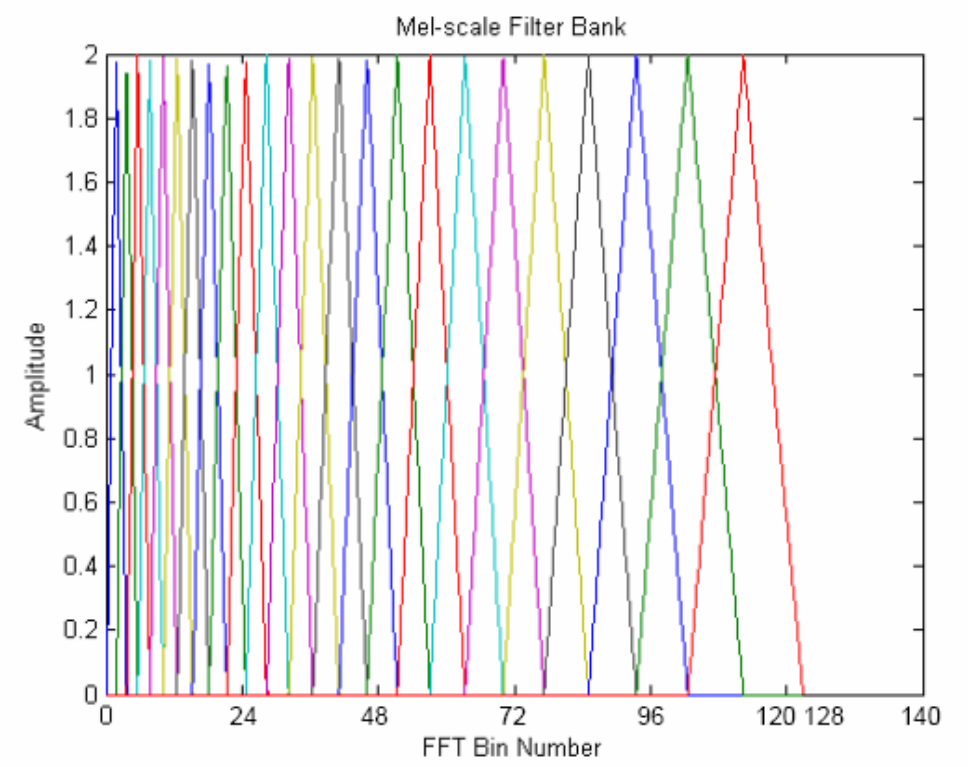

Figure 9. Mel Scale Triangular FIR Band-pass Filter Banks [Kurcan, 2006].

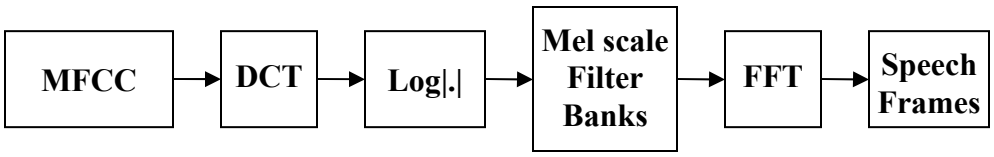

Figure 10. MFCC Feature Extraction Block Diagram

Mel Filter Bank index	Center Frequency (Hz)	BW (Hz)
1	100	100
2	200	100
3	300	100
4	400	100
5	500	100
6	600	100
7	700	100
8	800	100
9	900	100
10	1000	124
11	1149	160
12	1320	184
13	1516	211
14	1741	242
15	2000	278
16	2297	320
17	2639	367
18	3031	422
19	3482	484
20	4000	556
21	4595	639
22	5278	734
23	6063	843
24	6964	969

Table 1.2 Critical Band Filter Banks Based on Mel Scale [After Picone, 1993].

Mel filter cepstrum coefficients have been shown to be speaker independent and can be considered the most reliable features for use in speech recognition applications

[Davis, 1980], [Picone, 1993], [Kurcan, 2006]. However, Mel filter cepstrum coefficients are believed to be sensitive to noisy environments [Deller, 1993]. Thus, different techniques have been developed such as "high resolution spectral estimators or parametric fits of the cepstrum" to overcome noise related problems [Picone, 1993]. This thesis study uses an in-ear microphone which dramatically reduces environmental noise.

C. AUTOMATIC SPEECH RECOGNITION (ASR)

1. Automatic Speech Recognition

Automatic speech recognizers can be used to facilitate communication between humans and machines. Speech-based, human-machine interaction is demonstrated in several everyday applications, such as voice-mail systems in telephony, hands-free machine operations, communication interfaces for people with special abilities, dictation systems, and translation devices. ASR systems have been designed for different applications, in many areas, under a combination of restrictions, such as specific language and vocabulary, speaker dependency, noise-free environments and low talking rates, with excellent results. These existing systems are divided into three general categories [Deller, 1993]:

- Small vocabularies (10 to 100 words),
- Word-isolated ASR systems -- relatively small vocabularies (less than 10000),
- Continuous-speech ASR systems -- implemented in specific "areas" (from 1000 to 5000 words). The term "areas" refers to environments where the human-machine communication is restricted by its vocabulary, commands or subjects.

2. ASR - Problem Dimensions

ASR system design and performance is affected by several factors. Thus, the design of a reliable ASR system starts by examining the existence of these factors in the implementation "area." A general categorization of the factors that affect the design and performance of an ASR system follows [Deller, 1993]:

- Speaker dependency,
- Vocabulary size,
- Isolated words versus continuous speech,
- End-point detection problem,
- Vocabulary ambiguity and confusability,
- Noise characteristics of the environment,
- Linguistic constraints and knowledge.

Next, a brief description of the factors that affect the design and performance of an ASR system follows:

a. Speaker Dependency

Speech signal characteristics are dependent on the anatomy of a speaker's articulators. ASR systems usually use speakers for training and recognition performance evaluation. Two categories of ASR systems are widely used: Speech-dependent ASR systems, where the system is trained by one speaker and is used by this speaker only, and speaker-independent ASR systems, where a group of speakers is used for training, and the same group or a different group uses the system. Obviously, the former category is easier to implement than the latter, with better associated recognition results. This study considers a speaker-independent ASR system. In this effort, several speakers with different speech characteristics were used during the training process, to produce a robust system.

b. Vocabulary Size

The vocabulary size is an important parameter in the design and performance evaluation of an ASR system. The larger the vocabulary used, the more difficult it is to design the ASR system. Furthermore, as the size of the vocabulary increases, ASR performances typically decrease, mainly due to the effects of other factors, such as the acoustic confusability and ambiguity, and linguistic constraints. In addition, a large vocabulary requires a large amount of computational resources for real-time implementations. An ASR system's vocabulary is categorized as small (1 to 99 words), medium (100 to 999 words) or large (more than 1000 words). The boundaries of this categorization tend to change as systems of larger sizes (200,000 words) are introduced. This study uses a small vocabulary consisting of seven words which represent a minimum set of commands required for a machine control application.

c. Isolated Words versus Continuous Speech

Another consideration in the design and performance evaluation of an ASR system is the way utterances (commonly words or phonemes) are used during recognizer training and operation. The existence of "silence" between words characterizes *Isolated-Word Recognition* (IWR) systems. Silence, in this context, is the time interval between two consecutive words when the recognizer assumes that no speech

is present. Typically, an interval of 200 msec is the minimum requirement. Training of IWR systems involves multiple recordings of each isolated word from one or more speakers (depending on the degree of the ASR system's speaker-dependency). The recognition phase in IWR systems demands clear separation of words from one another by a "silence" interval. IWR systems are the simplest form of recognizer, usually achieving high performance, which is mainly dependent on an *end-point detection* scheme. This specific task is discussed further in Chapter V.

Continuous-speech recognition does not require a "silence" period between utterances, and the recognizer must be able to perceive speech at a natural human rate, with a minimum of constraints. This specification implies that such systems must work efficiently with coarticulation and other speech characteristics, such as intra- and inter-word articulation. Advanced end-point detection algorithms are used in CSR to account for variable boundaries issues. Furthermore, in some cases, a smaller utterance unit is used, such as a phone or a diphthong, instead of a separate word. In these cases, the training phase is more complicated and is focused on the inclusion of as many different utterances as possible. Moreover, linguistic restrictions help these systems perceive speech even when all utterances of a given sentence are not recognized, by using advanced cognition algorithms. CSR systems are obviously more complicated than IWR systems and usually demonstrate lower performance. However, CSR systems are widely used due to the fact that they seem to be a more "natural" speech communication interface for users.

d. End-point Detection Problem

End-point detection schemes play a critical role, especially in IWR systems. They are used by recognizers to find word boundaries and distinguish them from noise and "silence". Recognizer performance is degraded when such algorithms work inefficiently. Different approaches have been implemented for end-point detection, mostly based on zero-crossing and energy related measures. However, this is a challenging problem, particularly due to low energy phonemes, like fricatives, found at word beginnings or ends. To overcome this problem, various methods have been developed and are discussed in Chapter V.

e. Vocabulary Acoustic Ambiguity and Confusability

Acoustically ambiguous words are those with similar or homophonic spellings, such as "know" or "no," and "two" "to" or "too" [Deller, 1993], and are difficult to distinguish acoustically by their features during speech processing. Confusability describes the phenomenon where, given two words, one word might be incorrectly perceived as the other by a recognizer, mainly because they include the same phonemes. For instance, the alphabet letters "B," "C," "D," "E," "G," "P," "T," and "V." as pronounced, consist of a weak different consonant and the same voiced vowel. Their recognition depends only on the recognizer's ability to distinguish these weak consonant phonemes, a task which is not always easy, especially when noise is present. Generally, these two problems are present in continuous-speech ASR systems with large vocabularies. Neither problem is inherent in the words used for this study.

f. Noise Characteristics of the Environment

Different types of noise, such as measurement noise, processing noise and environmental noise, can affect an ASR system's operation. Measurement and processing noise are produced by the ASR's electronic components responsible for speech capturing and processing, such as the microphone, the microphone preamplifier, the analog to digital converter and the power supply. A special effort must be made during the design of this part of an ASR system to eliminate such types of noise. Additive environmental noise, a complicated type of noise, is defined as the sum of the noise signals captured by a microphone concatenated in the speech signal. This type of noise signals can be originated from other speakers in the room, by office or road noise, by the noise existing in the operation fields or by the speaker himself, with lip smacks, breath noises, pops, clicks, coughs or sneezes [Deller, 1993]. To overcome this type of noise, ASR systems use speech enhancement techniques that are based on the fact that in most cases, the noise is uncorrelated with the actual speech signal. Short-term spectral subtraction, Wiener filtering and adaptive noise canceling are among the most popular techniques in mitigating this type of disturbance.

A different approach to suppressing additive noise is to use microphones which are able to minimize the noise impact from the collected speech signal, as is the case for throat and ear microphones. The latter is used in this study and its characteristics are presented in Chapter III.

g. Linguistic Constraints and Knowledge

Linguistic constraints are the language rules which define the possible ways in which basic language units can be combined to form meaningful speech. Phonemes are considered the basic units of spoken language. In English, 42 phonemes are combined to form words according to *lexical* constraints, while words are combined according to *syntactic* rules to form sentences. Lexical and syntactic rules are parts of the language *grammar*. Moreover, *specific rules* guiding the combination of words and sentences in a meaningful and understandable way constitute another set of possible constraints. All constraints must be programmed in an ASR system (especially CSR systems) to increase performance. The basic idea is that the system can recognize a word even though it is not able to recognize the associated phonemes. Additionally, these systems can recognize a sentence from its context relative to neighboring sentences by applying linguistic constraints and rejecting all impossible combinations. Linguistic constraints are usually not used in small vocabulary IWR systems due to their additional complexity.

Next we briefly describe the speech features quantization process which is an intermediate step in ASR processing, usually applied before speech in a recognizer either for training or recognition.

3. Speech Features Quantization

Automatic speech recognition is generally a classification problem dealing with the classification of a sequence of speech features such as MFCC parameters, derived from speech processing procedures. Vector quantization schemes translate continuous-amplitude features vectors into discrete symbols (codes), removing all possible redundancy. These discrete codes correspond to different clusters in regions near a center (centroid).

Quantization introduces distortion which is the measurement of the distance between the feature vector and its quantized representation. Distortion decreases as the number of code vectors available increases. Unfortunately, as the number of code vectors increases, complexity and the computational demand of the system increases. This study uses 12 speech features and a 128-cluster codebook.

The K-means algorithm is used to obtain the codebooks in this study. It is a widely used method in speech recognition resulting in low distortion [Deller, 2000]. The K-means algorithm, where "K" refers to the number of associated clusters, is based on distortion minimization. Beginning from an arbitrary set of code vectors, the algorithm is fed iteratively with speech features vectors from the training samples. The algorithm applies the "nearest neighbor condition" and the centroid condition iteratively, until a termination criterion is satisfied" [Theodoridis, 2003]. The MATLAB implementation of this algorithm used in this study is taken from [Kurcan, 2006].

The resultant codebook is translated into hexadecimal format and hard-coded in the microprocessor's program memory. The classification procedure is done every time a new speech features vector is obtained from the microprocessor by calculating the Euclidean distance between this vector and each of the codebook vectors. The cluster with the minimum distance is chosen as the discrete symbol representation of the feature vector.

Next we briefly describe the Discrete Hidden Markov Model approach which is used as the speech recognizer.

4. Discrete Hidden Markov Model

Methods used to address the speech recognition problem can be distinguished into two categories: *dynamic time warping* (DTW) and *stochastic* (or *structural*) methods. The DTW method is a deterministic approach based on template matching. Speech feature warping (stretch or compression) in time, accomplished by "dynamic programming," is used to form a speech observation features sequence in such a way that it is comparable to a set of reference speech features sequences. Comparisons are made and results are evaluated to find a best match, concluding in a recognition result. Time warping is necessary due to the variability of speech utterances. DTW is a

straightforward method that has been implemented in VLSI circuit chips offering a solution particularly well-suited for real-time ASR systems.

The challenges associated with DTW method are the amount of memory needed to store utterance references (words in most cases), and the increasing computational effort needed for comparisons as the number of references increases. Furthermore, speech variability due to coarticulation, speaker variability, and noise, reduce the DTW algorithm recognition performance, limiting its implementation to specific systems (speaker-dependent, relatively small vocabulary) [Deller, 1993; Bahl et al., 1984].

Stochastic methods were introduced in speech recognition applications to overcome the speech variability problem. The most researched method in this category is the *Hidden Markov Model* (HMM) approach. To define HMMs we first define a general Markov Model as "a finite-state automaton with stochastic transitions (that is, for which each transition has an associated probability) in which the sequence of state is a Markov chain" [Gold, 2000]. Figure 11 shows a two-state Markov model with stochastic transition matrix A and initial probabilities $P(q)$ given as follows:

$$A = \begin{pmatrix} 0.5 & 0.7 \\ 0.5 & 0.3 \end{pmatrix}, \qquad P(q) = \begin{bmatrix} 0.3 \\ 0.8 \end{bmatrix}. \tag{2.3.1}$$

The probability of any observation sequence $Q = (q_1, q_2, \ldots, q_L)$ is given by the equation:

$$P(Q) = P(q_1)\prod_{i=2}^{L} P(q_i \mid q_{i-1}, q_{i-2}, \ldots, q_1). \tag{2.3.2}$$

Applying the first-order Markov chain assumption, where the probability of being in a particular state depends only on the previous state, Equation (2.3.2) becomes:

$$P(Q) = P(q_1)\prod_{i=2}^{L} P(q_i \mid q_{i-1}). \tag{2.3.3}$$

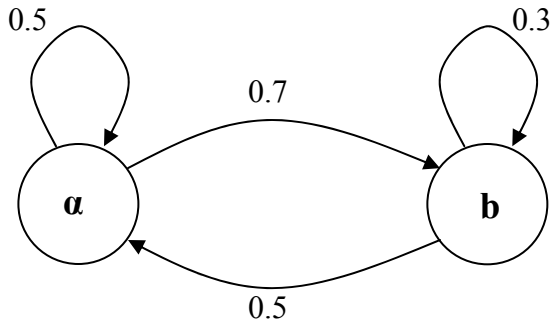

Figure 11. Two-State Markov Model

From Equation (2.3.3) we conclude that the probability of any sequence Q occurring can be found by multiplying the initial probability of the first state in the sequence by the transition probabilities to move along the sequence's states. In the example of Figure 11 with the matrix transition and initial probabilities of Equation (2.3.1), the probability of the observation sequence $Q = (a, a, b, a)$ is equal to:

$$\begin{aligned} P(Q) &= P(a)\cdot\left[P(a\,|\,a)P(b\,|\,a)P(a\,|\,b)\right] \\ &= 0.3\cdot 0.5\cdot 0.7\cdot 0.5 \\ &= 0.0525. \end{aligned} \tag{2.3.4}$$

Thus, a Markov Model based on a first-order Markov chain can be completely described (meaning that we can find the probability of any observation sequence in the model) by the initial probabilities $P(q_i)$ and the transition probabilities $P(q_i|q_{i-1})$. Note that even if intermediate states are unknown in an observation sequence, a probability can be obtained as the summation of the probabilities of all the possible sequence combinations. For instance, the probability of the observation sequence Q=(axb) in the model of Figure 11 is equal to:

$$\begin{aligned} P(Q) &= P(aab) + P(abb) \\ &= P(a)\left[P(a\,|\,a)P(b\,|\,a) + P(b\,|\,a)P(b\,|\,b)\right] \\ &= 0.168. \end{aligned} \tag{2.3.5}$$

A Hidden Markov model (HMM) is a type of Markov model whose output is a probabilistic density function. This means that the produced state sequence of a HMM is hidden for any observed output sequence. This type of model can be used to represent a

speech utterance such as word, sub-word or phoneme. Each of these utterances is modeled by a different HMM. The observations can correspond to speech feature vectors. In addition, a *discrete hidden Markov model* (DHMM) is produced when these feature vectors are represented by discrete codes (by applying vector quantization). Figure 12 shows a typical 5-state DHMM.

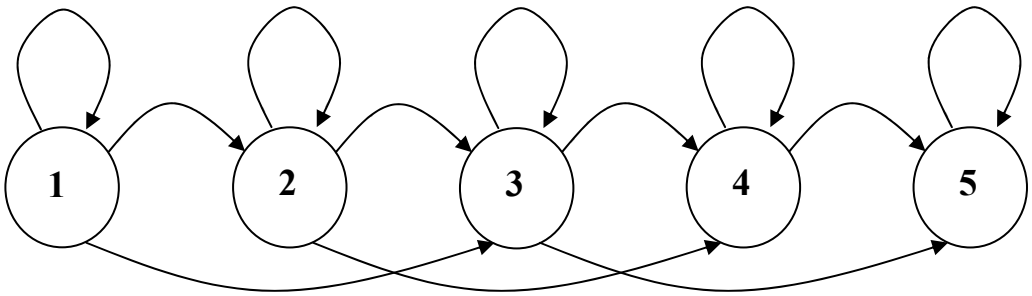

Figure 12. Typical 5-state DHMM

A DHMM can be fully described by the compact notation $\lambda=(A,B,\lambda)$ with: [Kurcan, 2006]:

- S, number of hidden states in the model,
- K, number of distinct observation symbols,
- The state transition probability distribution, $A=\{a(i\,|\,j)\}$, where $a(i\,|\,j)=P(q_i=i\,|\,q_{i-1}=j)$ and $1\le i,j\le S$,
- The observation symbol probability distribution, $B=\{b(k\,|\,i)\}$, where $b(k\,|\,i)=P(o_t=k\,|\,q_t=i)$, $1\le i\le S$, $1\le k\le K$, where o_t is the observation at time t,
- The initial state distribution, $\pi=\{P(q_1=i)\}$, where $1\le i\le S$.

5. DHMM Training and Recognition

Training and recognition are two basic problems addressed by a DHMM. The training problem seeks to determine how the HMM may model each speech utterance. From a mathematical point of view, the training problem comes down to estimating the model's transition matrix A, the observation probability matrix B, and the initial state

distribution vector π. The recognition problem focuses on estimating the likelihood of a speech utterance observation, produced by the model.

a. Baum-Welch DHMM Training Algorithm

Two methods are widely used to solve the training problem: the *Viterbi algorithm* and the *Baum-Welch (B-W) algorithm,* also known as the *Forward-Backward algorithm* [Baum, 1966; Baum 1970]. The later is used in this study, as implemented by Kurcan in its scaling version for multiple observations, due to its superior recognition performance [Rabiner, 1993; Deller, 2000; Kurcan, 2006]. With this algorithm, given a DHMM model $\lambda(A,B,\pi)$, forward and backward iterative computations are used to optimize the parameters of an estimated model $\hat{\lambda}(A,B,\pi)$ so that the likelihood of an observation *O* (speech utterance) occurrence can be maximized. Specific details about the implementation of the scaling multiple observations version of the B-W algorithm may be found in [Kurcan, 2006, Chapter IV.C.].

b. The Recognition Problem

The recognition problem, using a trained DHMM (meaning that parameters *A*, *B*, and π are known for all the models), is a scoring problem among the available models for each speech utterance. The quantized observation sequence is loaded into each model, from which a likelihood output is produced. The model with the greatest likelihood indicates the recognized speech utterance. The real-time implementation of this procedure is detailed in Chapter VIII.

An ASR system performance using DHMM mostly depends on the following two items: the specific speech utterance to be modeled and the number of hidden states required in the DHMM. Two different approaches have been used in the ASR literature to address the former question. First, words are used as speech utterances to be modeled by the DHMM. This approach is appropriate for small-vocabulary IWR systems, and thus implemented in this study. Second, sub-words such as phonemes and diphthongs are used as speech utterances to be modeled. This approach is used mainly in large vocabulary ASR and CSR systems where modeling with words would be inefficient with respect to memory allocation requirements (to store models) and computational effort.

The number of hidden states is the other crucial decision that must be made during the design of an ASR system. The larger the number of hidden states used in a DHMM, the greater the speech variability that can be modeled and the greater the computational effort required to calculate likelihoods. Unfortunately, there is no rule to assist in this decision. Generally, experimental results are used to suggest the number of hidden states in a DHMM [Deller, 2000]. In this study we use Kurcan's research and 8-state DHMM implementation [Kurcan, 2006].

D. REAL-TIME ASR SYSTEM CONSIDERATIONS

This study deals with the hardware implementation of a theoretical ASR model developed by Kurcan [Kurcan, 2006]. This implementation uses a microprocessor for a complete ASR system for the human-machine control and command interface. Such real-time systems differ from theoretical off-line models, even when the circuitry used to capture speech is the same. As a result, issues dealing with timing, processing resources and arithmetic precision must be considered.

1. Timing

Timing is the most important consideration for a real-time ASR system implementation. As mentioned previously, speech is captured and divided into overlapping frames prior to any processing. Then, short-term processing techniques are applied in each frame for end-point detection and feature extraction. The real-time system must have the computational power to complete these steps before the next frame is captured. Thus, compromises must be made between the maximum duration of a speech frame, invoking stationarity considerations and the time required for the system to complete frame processing.

Furthermore, the time needed to obtain recognition results may introduce delays between spoken words. Large delays demand a speaker to be silent for large periods of time, making the system inefficient. Thus, we designed our IWR system in such a way so that vector quantization and recognition procedures are conducted on a per-frame basis. The recognition result is available immediately after the current word's last frame processing is finished and before the first frame of the next word is captured.

2. Hardware Resources

Another critical issue is the availability of system resources such as memory, digital signal processing cores (DSP cores) and input/output (I/O) ports. Achieving the "right" timing specified for a real-time ASR system is not only dependent on the microprocessor's clock rate. Temporal (random access memory RAM) and static (Read only memory (ROM), electrical Erasable Programmable ROM (EEPROM), FLASH memory) memory devices are needed to store the temporal results of speech-frame processing and related coefficients. All these issues must be examined relative to system size and cost, and are dependent on the ASR application.

The complexity of digital signal processing algorithms, such as the Fast Fourier Transforms (FFT), requires a large number of computations and would result in large delays if these tasks were accomplished directly by the central processor unit (CPU). However, this problem is solved by the use of a digital signal processing core which is able to perform complicated mathematical operations, independent of the CPU, and usually in just one instruction cycle. A digital signal processing core is found in advanced microprocessor systems dedicated to signal processing applications. This type of microprocessors was chosen for our real-time ASR system implementation.

The type and number of available system I/O ports can affect the ASR system design. As a human-machine interface, the ASR system must first receive a signal of interest in a proper way, and second, send the associated commands to a machine. In addition, such a system must be programmable, and, especially for the HMM implementation, trainable. Thus the minimum requirements for I/O ports are one analog input for the analog to digital converter (if the conversion occurs inside the microprocessor), and up to three digital bidirectional ports for communication with the machine, the external device programming and the speech capturing device.

3. Software Analysis, Design and Implementation

Software analysis, design and implementation are the basic steps required to transition the ASR theoretical processing model into a microprocessor-based system. Analysis is the most critical of the three, because it is the step where the theoretical and technical information of the ASR system are combined to produce proper processing algorithms. The result of this step is a complete analysis block diagram which describes

the signal flow from the point of capturing a signal of interest to the delivery of final commands to the machine. Design is the step where every “black box” of the analysis block diagram is divided into its components, and software and hardware decisions are made for their implementation, such as programming language, memory buffer size, DSP core usage and I/O port requirements. Finally, in the implementation step, the design components are transformed into microprocessor programs and are loaded into memory. This step also incorporates all the required tests for system evaluation.

Next, we present the characteristics of the hardware used for the IWR system implementation.

III. IWR SYSTEM HARDWARE

A Real-time IWR system may be divided into functional areas such as the speech capturing system, and the speech DSP processing unit. The former may consist of the microphone, the pre-amplifier, the filter, and the analog to digital (A/D) converter. Figure 13 shows the block diagram of such a system and the signal flow among its components.

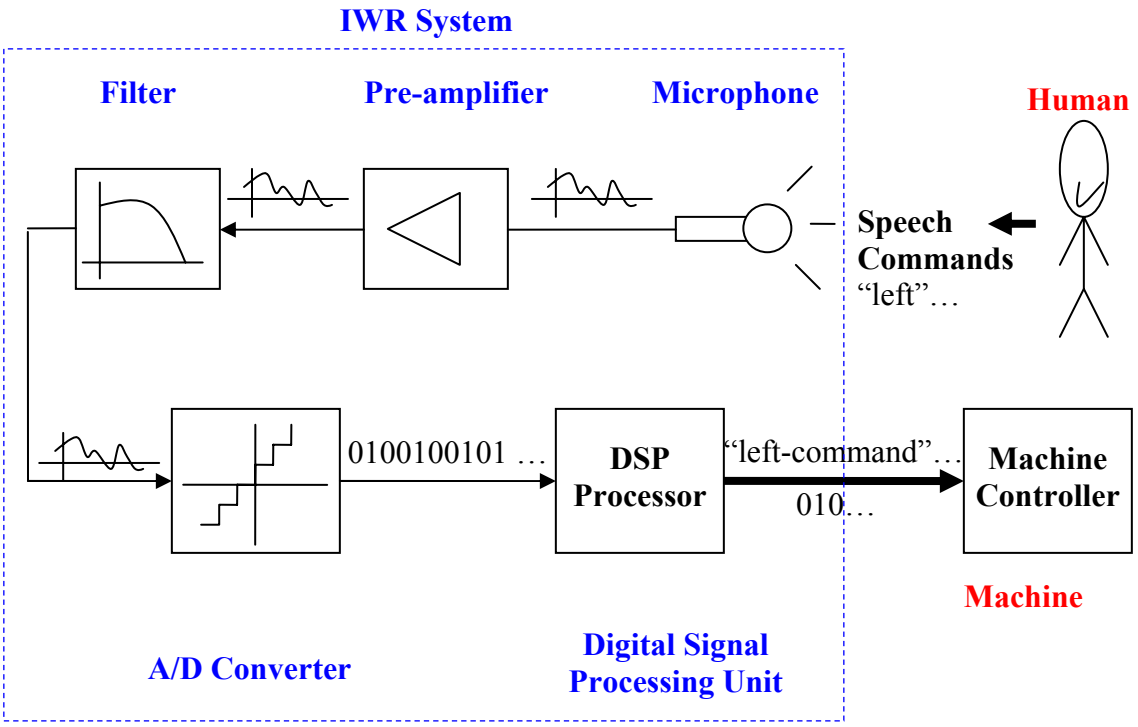

Figure 13. Block Diagram of a Real-time IWR Human-machine Interface for Robotic Control

The objective of this study is to examine the real-time implementation of a 7-word IWR system as a human-machine interface for robotic control. This implementation is based on inexpensive commercial COTS components. First, we selected an ear microphone readily found in the commercial sector; the Ear Bone Microphone - XEM98D from IXCESSORY [IXCESSORY, 2007]. Second, we selected an audio codec that includes a pre-amplifier, amplifier, filter, and 15bit A/D converter contained in a compact case. The model is the Si3000 audio codec from Silicon Laboratories. Finally,

the dsPIC33FJ256GP710 microprocessor from Microchip is used as the DSP processor unit. Next we describe the specifications and the architecture of our system's components.

A. SPEECH CAPTURING SYSTEM

The speech capturing system is responsible for transforming speech from air pressure into an electrical signal. Usually an open-air microphone placed in front of a subject's mouth is used and results in good recognition performance in a noise-free environment. However these systems yield poor performance in noisy environments. Different methods have been developed and implemented in ASR systems to minimize the noise problem. These include placing the microphone in a different location; for example, the microphone may be placed at the throat level instead of in front of the mouth, or active and passive microphones may be combined [Zhang, 2004; Graciarena, 2003]. Westerlund et al. proposed an Active Noise Control (ANC) headset equipped with an in-ear microphone and ear muffs. Commercial applications of this principle can be found in the Quiet Pro Headset [Westerlund, 2003; Nacre, 2007].

Vaidyanathan et al. investigated an in-ear noise shield air pressure sensor, capable of capturing tongue movements [Vaidyanathan, 2004]. Later, based on this study, Kurcan designed and evaluated a 7-words off-line IWR system using a noise shield in-ear microphone to capture the speech signal [Kurcan, 2006]. Motivated by the latter two studies, we extended this research by attempting to implement a real-time 7-words IWR system using an in-ear microphone.

Figure 14 shows the commercially available ear microphone used in this study. This microphone differs from that used by Kurcan, but has comparable signal capturing and noise shielding characteristics.

We used the compact Si3000 audio codec by Silicon Laboratories for microphone signal amplification, filtering, and sampling. This system includes a microphone pre-amplifier, a FIR digital low-pass filter, and an A/D converter. The SI3000 audio codec device is programmable, allowing changes in the systems amplifier gain and A/D sampling frequency. A block diagram of the Si3000 audio codec is presented in Figure 15 [After Si3000 Data sheet, 2000].

Figure 14. Ear-microphone Used in this Study

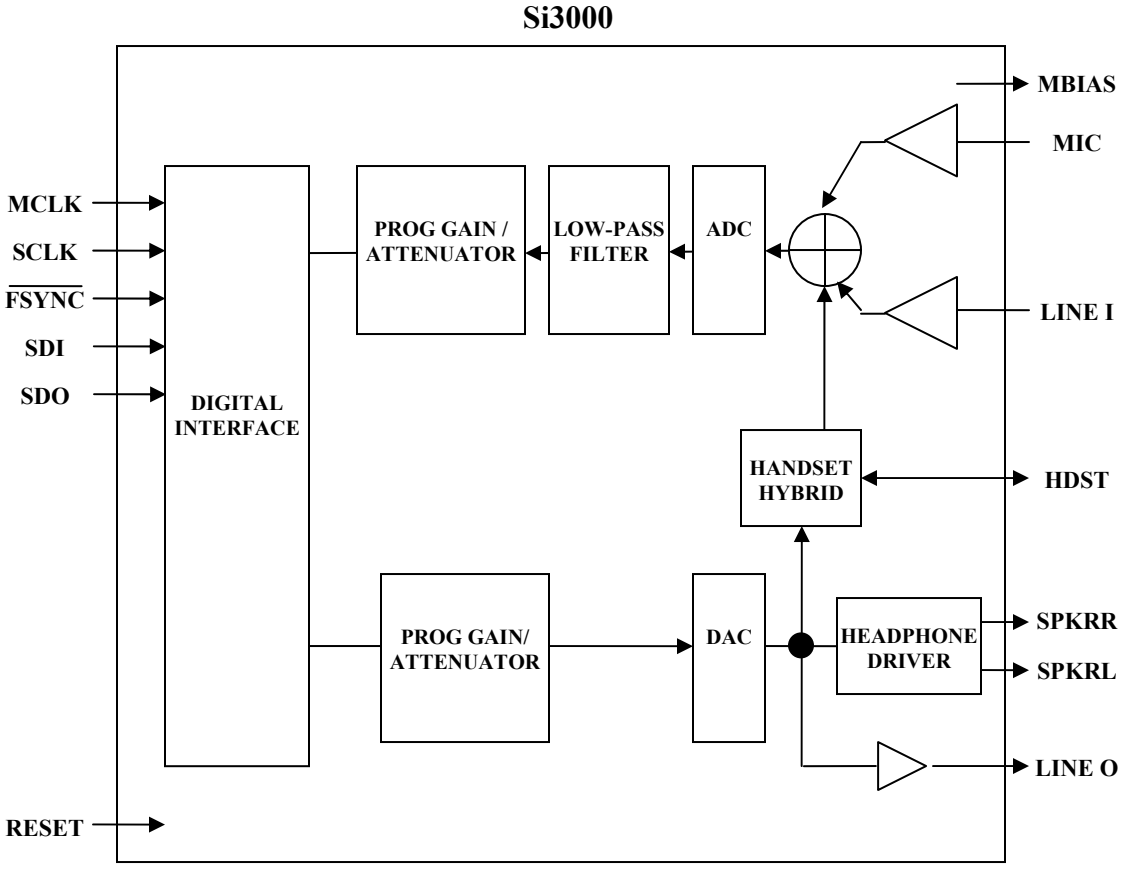

Figure 15. Si3000 Audio Codec Block Diagram [After Si3000 Data sheet, 2000]

The Si3000's microphone pre-amplifier can be programmed for four different gains: 0dB, 10dB 20dB and 30 dB. A 20dB gain was selected for the study empirically as speech captured by the ear-microphone is weaker than that captured by a microphone placed in front of the mouth.

The Si3000 contains two types of low-pass filters: a finite impulse response (FIR) and an infinite impulse response (IIR) filter. The pass-band for these two filters is 0-3600 Hz. Note that speech captured by the in-ear microphone is already been low-pass filtered by the face bones and cheeks, and most of its energy is found in frequencies below 2500 Hz [Kurcan, 2006]. Thus, the codec's low-pass filter is adequate for the spectrum characteristics of the ear-microphone's captured speech signal. The FIR filter characteristics, as taken from the Si3000 datasheet, are shown in Figure 16.

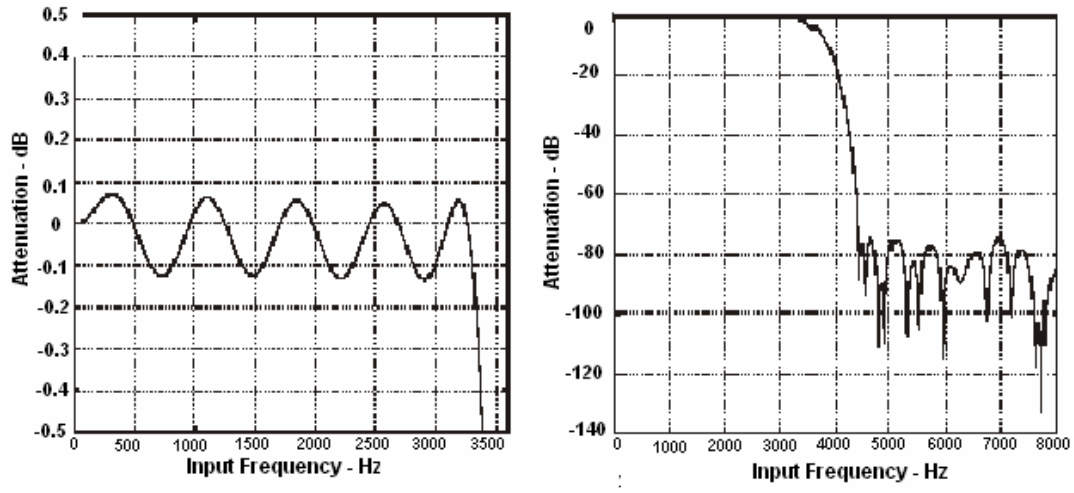

Figure 16. Audio Codec FIR Filter Pass-band Ripple and Frequency Response

The codec's A/D converter was used instead of the microcontroller's due to its better quantization performance. The codec's A/D uses 32,768 quantization levels (2^{15}), while the microprocessor's A/D uses 1,024 (2^{10}) quantization levels. Moreover, the compact structure of the codec introduces less processing noise in the signal path from the microphone to the A/D converter. The sampling frequency F_s is 8 KHz, since the speech signal does not exceed 3600 Hz (less than $F_s/2$).

The Si3000 codec's operation is controlled by nine programmable registers. The meaning of each register and associated values (as programmed in this study) are given in Appendix B. Briefly, the Si3000 is programmed for a microphone input with a 20 db pre-amplifier gain, an 8 KHz A/D sampling frequency, a low-pass FIR filter and assigned as slave in its communication with the microprocessor's DCI device. Figure 17 shows the complete capturing system implemented in this study.

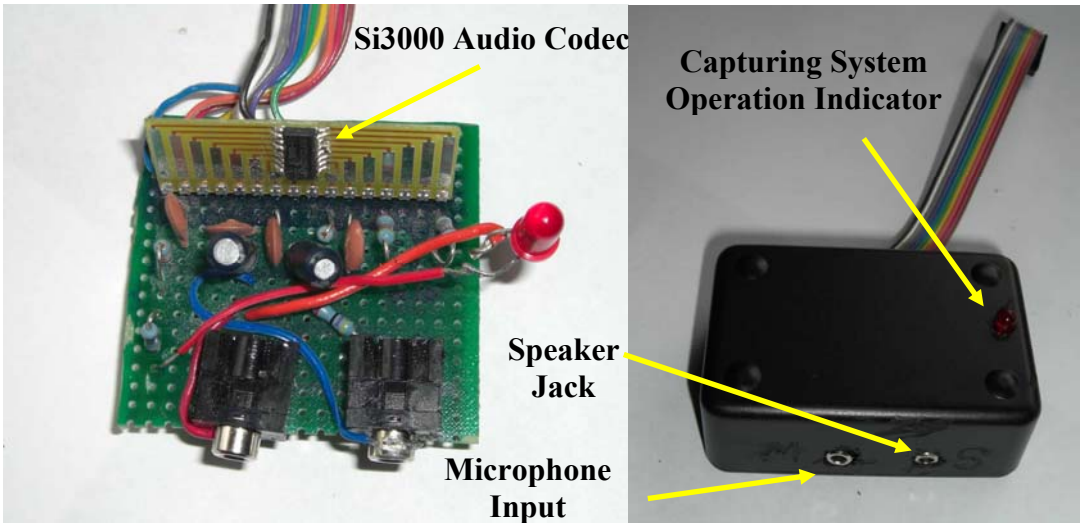

Figure 17. Capturing System

Next we briefly describe the dsPIC33FJ256GP710 Microchip's microprocessor used as digital signal processing unit.

B. DSP PROCESSING UNIT

The choice of the DSP processing unit is critical to a real-time IWR system implementation. Speech digital signal processing includes complex and computationally demanding mathematical operations such as FFT and matrix multiplications. Furthermore, real-time DSP requires these operations to be executed within the time allocated to the capture of the next speech frame. In addition, as a human-machine robotic control interface, our system must be implemented in a small and portable fashion with low power consumption. This study is implemented using the Microchip's dsPIC33FJ256GP710 microcontroller which meets the majority of our system demands.

It is a low cost and power efficient microprocessor which requires few external components for its operation (power supply, external oscillator) leading to a very robust, yet small system.

The dsPIC33FJ256GP710 microprocessor contains a 16-bit "Harvard" architecture Central Processing Unit (CPU) and a DSP core running at a maximum clock rate of 40 mpics (million processor instruction cycles per second). It includes 16 general purpose registers, a data memory of up to 32 Kbytes and a program flash memory of up to 256 Kbytes. In addition, a variety of I/O modules are included such as a Data Converter Interface (DCI), Inter-integrated Circuit (I2C), and Universal Asynchronous Receiver Transmitter (UART). The DSP core employs one 40-bit Arithmetic Logic Unit (ALU), two 40-bit saturating accumulators, one high speed 17 bit multiplier and one 40-bit bidirectional barrel shifter.

The dsPIC33FJ256GP710 microcontroller uses assembly language. Special commands for DSP operations are available, making the processor more powerful and time efficient for complex and demanding computations. Commands such as "mac" execute multiplication and addition, and fetch the next operands in one processor cycle. All operations can be conducted in two different arithmetic formats: 16-bit integer and 1.15 fractional, as described in Appendix F. Furthermore, 32 bit integer and 1.31 fractional operations can be programmed. The block diagram of the dsPIC33FJ256GP710's CPU and DSP core are shown in Figure 18. The microcontroller's specifications are given in Appendix A.

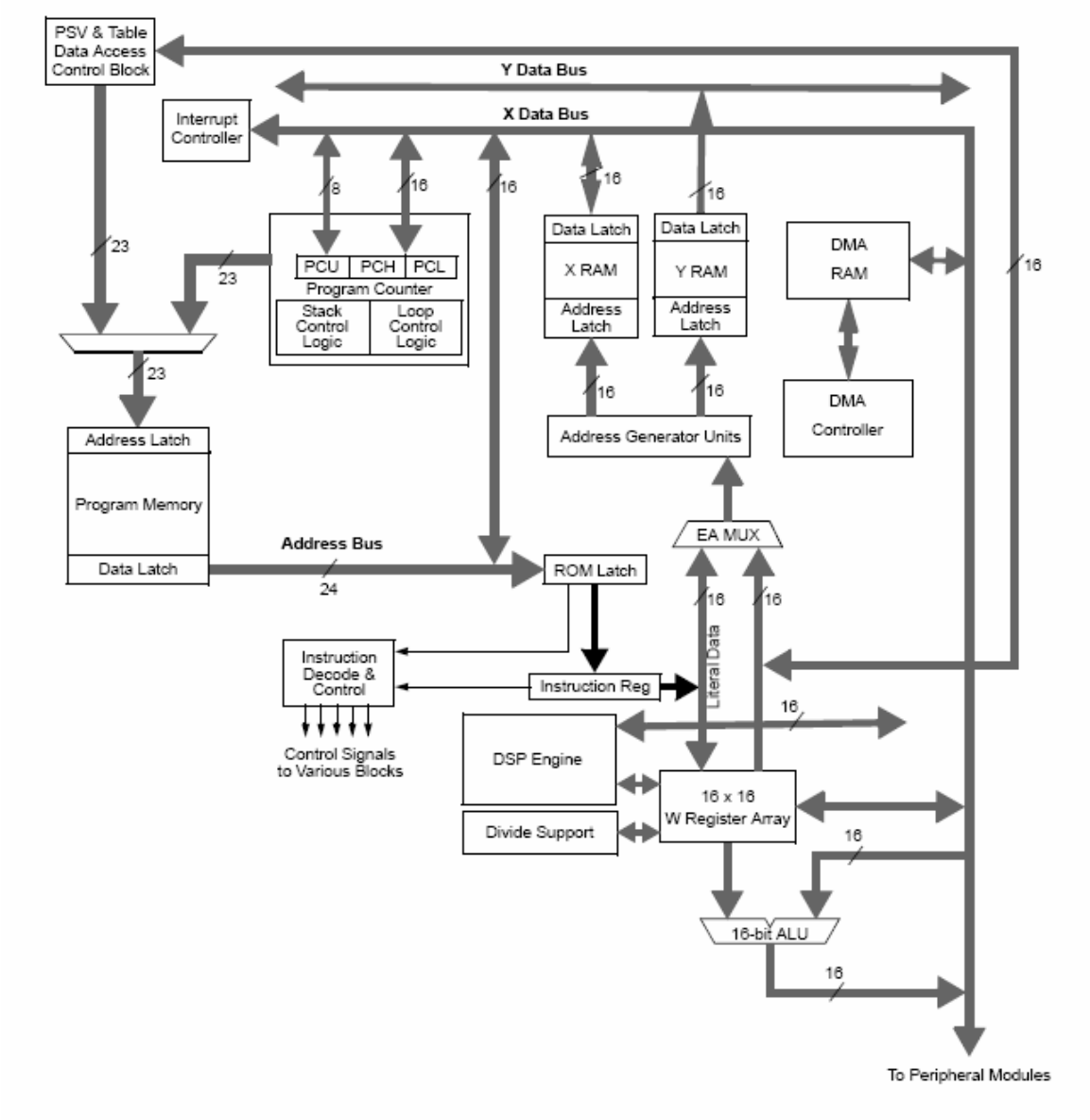

Figure 18. Microchip dsPIC33FJ256GP710's CPU and DSP Core Block Diagram [Microchip, 2006]

Next we present the demo-board used in this study for the complete implementation of the system.

C. SYSTEM HARDWARE COMPLETE IMPLEMENTATION

The development demo-board Explorer 16 of Microchip was chosen for the complete implementation of our system. The demo-board supports the dsPIC33FJ256GP710 microprocessor providing power supply, serial com port, LCD

display, four buttons, eight LEDs, and an area for developing custom applications. Furthermore, the board, in combination with the Microchip's ICD2 programmer, can be used for microprocessor's programming and real-time debugging. Figure 19 shows the Explorer 16 development demo-board.

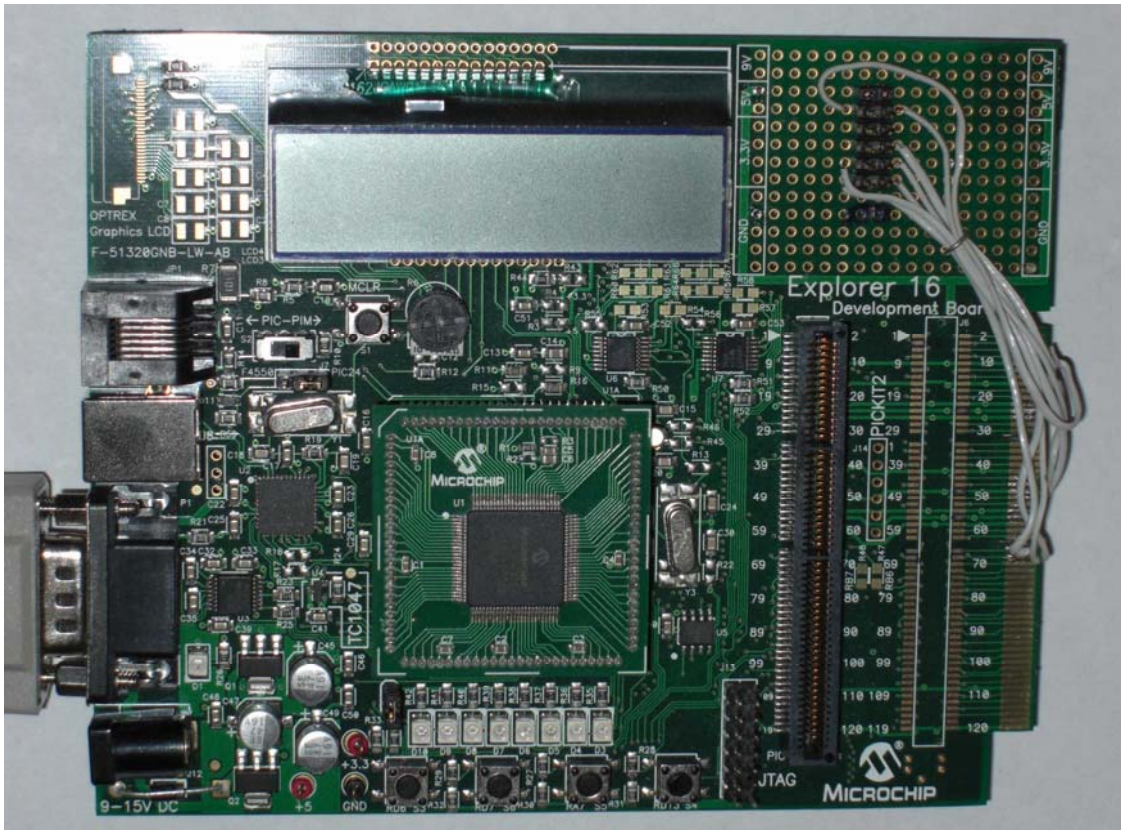

Figure 19. Explorer 16, Development Demo-board by Microchip

Next we discuss the real-time implementation's software design of the IWR system developed in this study.

IV. REAL-TIME IMPLEMENTATION

A. REAL-TIME IWR SYSTEM SETUP

Real-time digital signal processing systems are usually implemented in embedded microprocessor systems dedicated to a specific application. Several different processor platforms can be found commercially. In this thesis, the Microchip's dsPIC33FJ256GP710 microprocessor was chosen mainly due to its capabilities relative to its cost and development simplicity. All procedures were developed in the microprocessor's assembly language using Microchip's MPLAB V5 integrated data environment (IDE) and the ICD2 Microchip's programmer. In addition, the Explorer 16 Microchip's Development Demo Board was used for hardware and software evaluation. The microprocessor's I/O parts used in this study are the Data Converter Interface (DCI) and the Universal Asynchronous Receiver Transmitter (UART). The microprocessor's main features are listed in Appendix A.1, and Appendix A.2 provides a block diagram of the microprocessor, including its CPU, DSP Core, memory, and I/O devices. The only external components required for system operation is a 3.3V power supply, an oscillator circuit, an RS232 integrated interface, and the speech capturing system described earlier in Chapter III. Figure 20 shows the overall IWR system setup which has a total cost under $50 (including the microphone, when implemented separately of the demo-board Explorer 16).

Timing issues and hardware resources constraints are the two techniques challenges discussed earlier in Section II.D, which were faced and resolved in this study.

Timing issues, previously mentioned in Section II.D.1, were the result of the IWR system real-time implementation considered here. Specifically, end-point detection, frame feature extraction, frame feature vector quantization, and recognition operations needed to be executed by the microprocessor within a time period less than or equal to the time required for a speech frame to be captured. This constraint introduced timing issues which could have resulted in the loss of speech data without specific consideration.

Next, hardware resources constraints, discussed earlier in Section II.D.2, were due to the finite microprocessor's data memory capacity. For illustration purposes note that a

5-second long captured signal (including speech and silence period) with an A/D conversion of 8 KHz sampling frequency and 16 bits sample's binary representation, needs 78.125 Kbytes of data memory (RAM). This amount of memory can be found in expensive microprocessors. However, the low cost microprocessor used in this study has only 32 Kbytes of data memory.

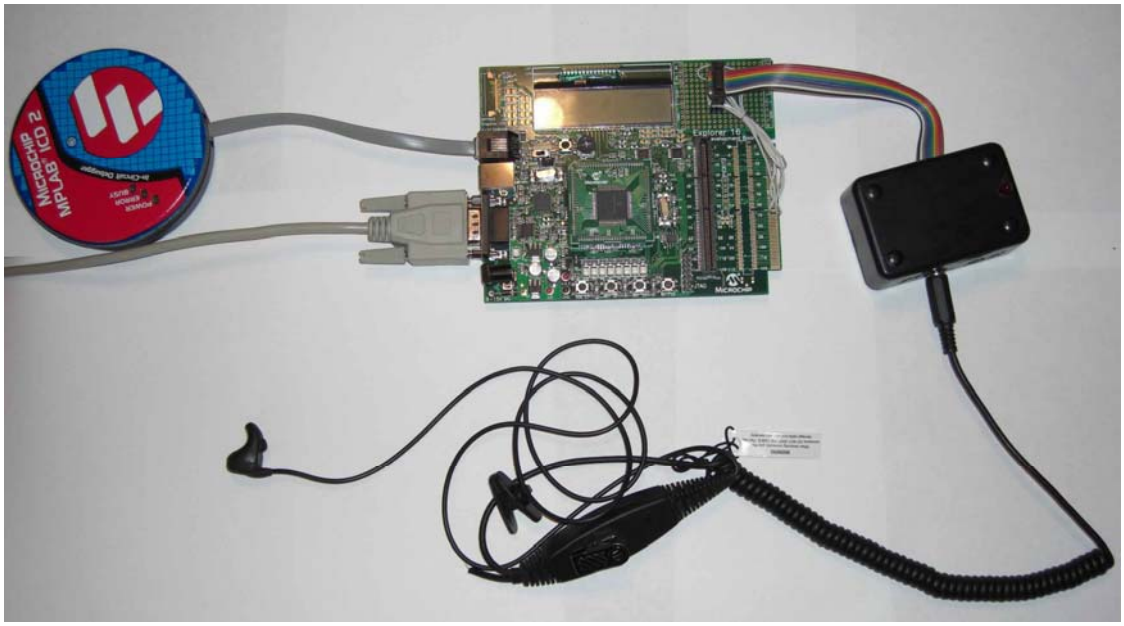

Figure 20. Real-time IWR System Setup

We addressed the first challenge by using three different programming approaches. First, we decreased the microprocessor Instruction Cycle Period to 0.3398 nsec (29.5 MHz). Second, we optimized the algorithm's assembly routines to include as few instruction cycles as possible. Finally, we used semaphores, flags and buffer pointers to control the program flow and avoid procedures to compete for processor's resources such as data memory.

The second challenge was resolved by storing only the part of the captured signal where speech is detected. Therefore, the required data buffer memory length for storing the spoken word was reduced to the maximum expected word's length, which was experimentally observed to be 5,520 samples or 690 msec (for 8 KHz sampling frequency). The A/D converter uses 16 bits to represent each sample (15 bits used for the

quantization level and the least significant bit is “don’t care”). Therefore, a data memory buffer of length 11,040 bytes (2x5,520 bytes) was the minimum requirement. In addition, we extended the buffer’s length by 960 additional bytes (480 samples or 60 msec) to insure that detected words would be completely stored into the buffer. Thus, the final data memory buffer’s length used in this study to record speech is 12,000 bytes. Note that the microprocessor’s data memory is aligned in data words using the “little indian” format meaning that each data word consists of two bytes (one byte is eight bits) where the least significant byte’s memory address is even.

In addition to the data memory buffer used to store the spoken word, other data buffers are used for storing the results of intermediate processing routines. The required length in bytes of these buffers was determined by two factors; the choice of the frame length, and the expected length in bytes of the results. A 256-sample frame was chosen in this study to implement the short-term speech processing for the end points detection task, feature extraction, and recognition. Figure 21 describes the framing process followed. Each frame is overlapped with the previous one by 53% or 136 samples. The following frame is captured after an additional 120 (256-136) new samples are recorded. The maximum number of 256-sample overlapping frames for each word is equal to 48 ((6,000-240)/120+1, where 6,000 is the total length of the spoken word’s data buffer, while the length of the first frame is equal to 240 samples, as explained in Chapter VI, and 120 is the number of additional samples to the 136 samples already recorded in the previous frame). The exact length of each data memory buffer used in this study is separately presented in combination with the routine that is used at that point. The total amount of data memory used in this study is 16,320 bytes, corresponding to 53% of the microprocessor’s total available data memory.

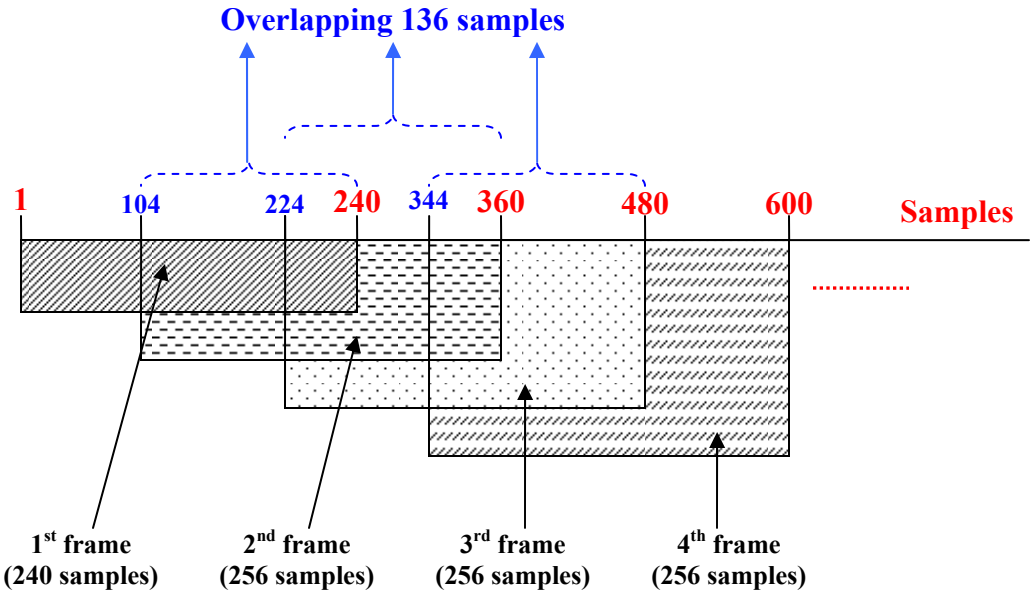

Figure 21. Framing Process of Capturing Speech Samples

Next we present the main program and the interrupts used in this study.

B. MAIN PROGRAM

The main program is the basic assembly routine used in the real-time implementation to control the execution of the whole program. It consists of four different parts: Microprocessor Initialization, Data Converter Interface Interrupt, Speech Processing-Recognition, and Robotic Control. Figure 22 depicts a block diagram of the main program.

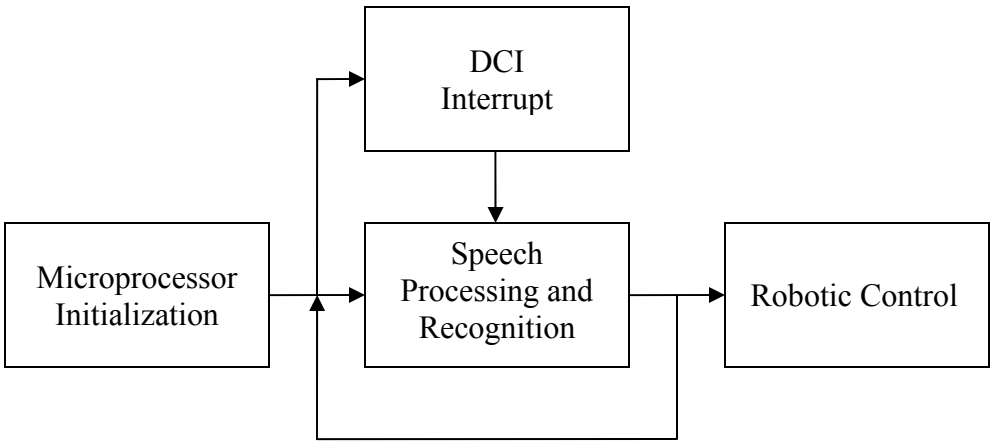

Figure 22. Block Diagram of the Real-time IWR system's Main Program Routine

1. Initialization Routine

The I/O devices (DCI, UART, I/O ports), the microprocessor's clock, the audio codec, and the semaphores, flags, and global variables used by the program are setup during the Microprocessor Initialization Routine. The DCI is initialized to produce an interrupt only when its 4-word receiver buffer is full. The DCI is programmed as the master, providing a clock signal to the audio codec. DCI and audio codec initialization parameters are listed in Appendix B. Ports D6 to D7 and ports A5 to A10 are initialized as input and outputs, respectively. Port D6 and D7 are associated with push buttons and ports A5-A10 are associated with six LEDs (light emitting diodes). The UART is set for 115,200 bps with 8-bit data, no parity bit and no flow control. The initial values for all flags, semaphores and global variables are listed in Appendix D. After initialization, the main program is set to a repeat loop, awaiting user activation of the DCI interrupt and the speech-processing and recognition steps by depressing button no. "1" in the microprocessor board [Section IX.A].

2. Data Converter Interface Interrupt

The Data Converter Interface Interrupt is used as the main synchronization source in data flow control, since the DCI controls the Audio Codec and consequently the speech samples. The DCI interrupt occurs when four samples have been captured from the codec and transmitted to the microprocessor. Thus, this interrupt occurs every 0.5

msec or 14,750 pics (processor's instruction cycles per second) with a 8 KHz sampling frequency. The following two different theoretical timers based on this DCI interrupt are used:

- Four samples timer or 14,750 pics (0.5 msec)
- Forty samples timer or 147,500 pics (5 msec)

The first timer is used to read the samples from the DCI buffer, to save the samples to the word buffer, to update the word buffer pointer, and to count the captured signal's samples. When 40 samples have been captured the "*Frame40CompleteFlag*" flag is set, triggering the second theoretical timer. The second timer is used for the activation of every other routine in the program.

3. Speech Processing-Recognition Program Part

The Speech Processing-Recognition part is represented by the "*main_loop*" routine. It includes five different flag-checkpoints which direct the program flow in four routines, as shown in Figure 23. These checkpoints and their effect on program flow are presented in their order of appearance in the *"main_loop"* routine:

- "*EndofBufferFlag*" checkpoint: the word buffer is full; the main program terminates and starts again from initialization when this flag is set. Else, the program continues its execution. The flag is controlled by the DCI interrupt which compares the number of captured samples with the total length of the spoken word's data buffer.
- "*Frame40CompleteFlag*" checkpoint: the program goes to the last checkpoint ignoring all the intermediate ones when this flag is cleared. Else, the program clears the "*Frame40CompleteFlag*" flag and goes to the next checkpoint. The flag can be set only by the DCI interrupt, when 40 speech samples have been captured.
- "*InitialThresholdFlag*" checkpoint: the program first starts threshold calculations as long this flag is not set, and all other routines are non active. The flag is set when the thresholds have been calculated and remains set until a new system initialization occurs. When this flag is set, the program is directed to the rest of the operations (normal_procedure) [Section V.A.3].
- "*FinalStartpointFlag*" checkpoint: the program redirects to the next checkpoint as the final start-point has been detected when this flag is set. Else, the program goes to the last checkpoint ignoring the intermediate one. The flag is cleared at the beginning of the main loop and is set by the start-point detection routine. It remains set until a new word is ready to be processed.
- "*EndPointDetHasRunFlag*" checkpoint: the program redirects to the feature extraction activation routine when this flag is set. Else, the program goes to the next checkpoint. The flag is set any time the end-point detection routine has run. It is used

as a synchronization flag for feature extraction described in Chapter VII. The flag is cleared immediately after the checkpoint.

- "*DCIDataAvailableFlag*" checkpoint: This is the last flag checkpoint of the main_loop routine. A valid recognition result is available and the program redirects to the robotic control part when the flag is set. Else, the program restarts the *main_loop* procedure.

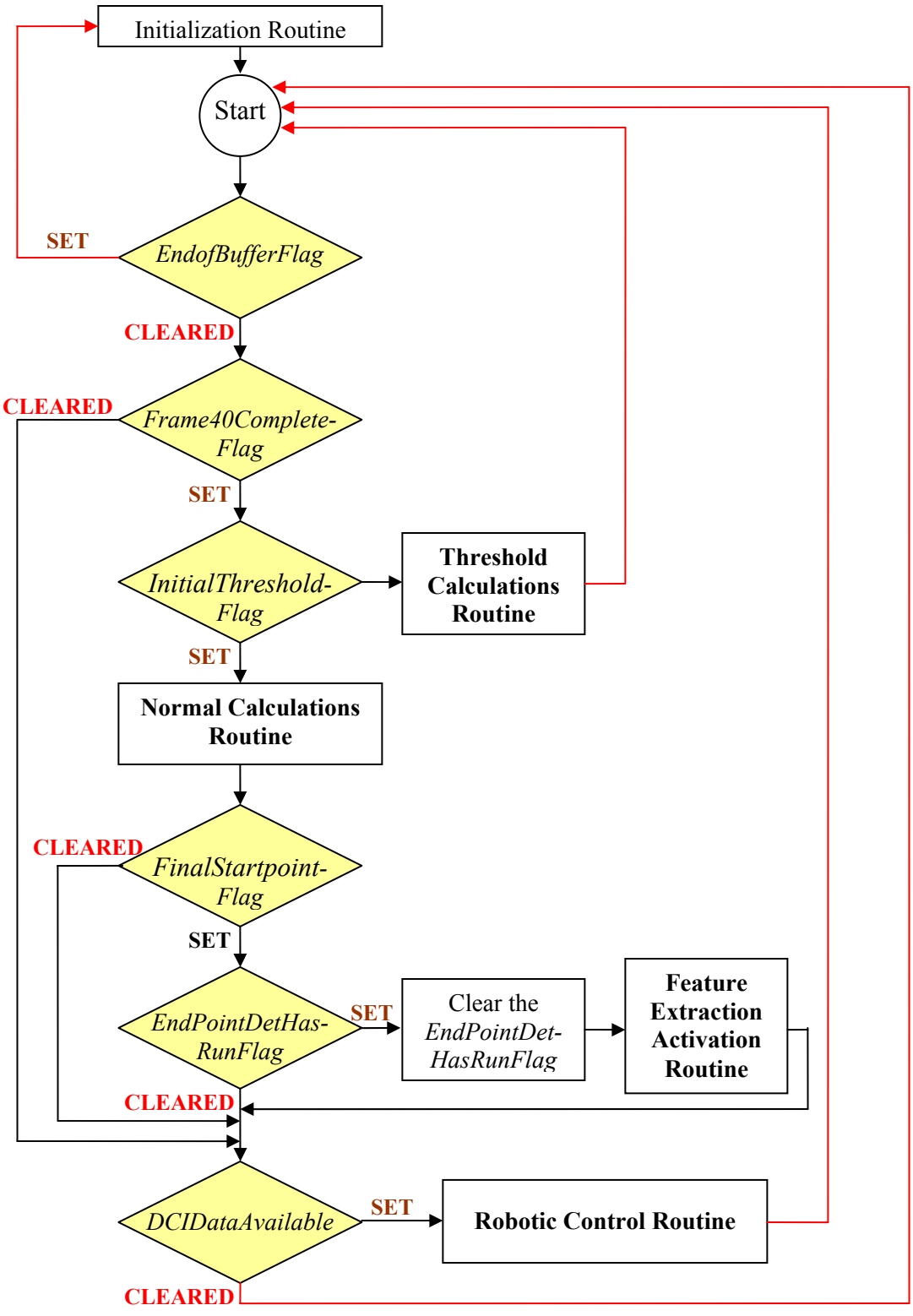

Figure 23. Block Diagram of "*Main_loop*" Routine

4. Robotic Control Program

The robotic control program part is responsible for the translation of the recognized word into a control machine command. We use four bits (2^4=16 combinations) for the binary representation of the commands corresponding to the 7-word vocabulary. In addition, two more binary numbers "0111" and "1111" are used to indicate that a recognition result is not available and that an error has occurred, respectively. Table 3.1 lists the binary control machine commands associated with the recognized words and indicators.

Command and Indication	Binary Representation
"up"	0001
"down"	0010
"left"	0011
"right"	0100
"kill"	0101
"pan"	0110
"move"	0111
No recognition	1001
Error	1111

Table 3.1 Binary Control Machine Commands in Respect with the Recognized Words and Indications

The UART port is used to communicate with the robot. The *UART_init* procedure initializes the UART port for transmission/reception at 115,200 bps, with eight data bit, no parity, and no flow control. After a command is sent, the program waits for a timeout and for the robot to acknowledge the command reception. The program repeats the transmission-reception section until an acknowledgment received or until a new 40-

sample frame is captured. In any case the program redirects to the *main_loop* start point continuing the reception-recognition part. The LED labeled no. "6" on the microprocessor board indicates a successful command reception by switching to an 'ON' state, and remains in an "OFF' state otherwise, as described in Section IX.A.1.

5. Error Identification Program Part

The main program also includes the error identification part which consists of four error traps:

- Oscillator Fail Error trap routine,
- Address Error trap routine,
- Math (Arithmetic) Error trap routine,
- Stack Error routine.

These routines trap the microprocessor every time the corresponding error occurs and is shown by illuminating a specific LED, described in Section IX.A. Note that the only way to exit an error trap is by resetting the microprocessor.

Next, we discuss the procedures used to solve the end points detection problem.

V. END-POINT DETECTION

A. END-POINT DETECTION ALGORITHM

1. Introduction

The end-point detection algorithm is based on the *Short-Term Energy (STE)* and the *Zero-Crossing (ZCR)* measure calculations of a given signal. The idea behind these approaches is based on the differences in energy and frequency between speech and silence's period (no speech). The captured signal consists of a silence period, a spoken word, and a silence period. Plotting the STE of the captured signal we observe that the energy where speech is present is larger than that found during silence periods. Thus, the spoken word's bounds can be determined as the points where the energy of the captured signal is greater than a threshold. The threshold is calculated as the possible maximum STE of the silence period. Furthermore, the ZCR measure is used to extend the bounds of the spoken word, in case where low energy but high frequency phonemes occur at the beginning or at the end of the spoken word.

Absolute "silence" cannot be achieved because the capturing device (microphone) introduces a level of processing noise even when no speech is present. Thus, we use an energy threshold to define "silence" periods. A signal is considered to be speech if its energy is above this threshold, and noise or "silence" when its energy falls under it. The STE for a frame f of length N of a speech signal $s(n)$ is given by:

$$E_f = \sum_{n=0}^{N-1} s^2(n) w(n), \tag{5.1.1}$$

where $w(n)$ is a window required for the validity of short-term signal processing [Section II.B.1]. A rectangular window is used in this process for simplicity.

The problem with the STE algorithm is that many words begin or end with weak unvoiced or voiced phonemes such as fricatives, whose energy is lower than the calculated thresholds. In these cases, the utterances are considered as "silence" and are ignored by the algorithm, introducing problems during subsequent steps of the recognition procedure. The Zero-Crossing (ZCR) measure was added to overcome this problem.

The Zero-Crossing measure is used by the end-point detection algorithm for refinement of the word's bounds as computed by the STE algorithm. The ZCR is a frequency measurement, representing the number of times a zero-mean signal changes sign during a frame. A high-frequency signal implies a high ZCR. The weak utterances at many word's ends are commonly of high frequency such as "t" at the end of the word "left." The initially calculated "silence's" ZCR (mean and variance) defines the ZCR threshold. Frames near the word's bounds as detected by the STE method with ZCR greater than this threshold are considered speech. The frame ZCR is given by the following formula:

$$ZCR = \frac{1}{N}\sum_{n=0}^{N-1}\frac{\left|sign\left(s\left(n\right)\right) - sign\left(s\left(n-1\right)\right)\right|}{2}. \qquad (5.1.2)$$

In this thesis, the STE and ZCR measure algorithms introduced by Rabiner and Sambur [Rabiner, 1975] and modified by Kurcan [Kurcan, 2006] are used. Further modifications have been applied to the algorithm for real-time microprocessor processing.

2. Threshold Calculations Procedure

The threshold calculations procedure is an algorithm which is run initially to obtain the noise level of the input speech line and to define the "silence" levels. It sets up the short-term energy minimum and maximum thresholds and the ZCR threshold used by the end points detection algorithm. The procedure is applied to the first 500 msec of the input signal, where only noise is assumed. The first 400 msec are ignored, while the last 100 msec are divided in frames of ten msec, overlapped by 50%. Thus, 19 overlapped frames are examined. The short-term energy and the zero-crossing measure are computed for each frame. These values are stored in a buffer and are used by the algorithm to compute the mean and standard deviation of the short-term energy and zero-crossing measure. The thresholds values are obtained by multiplying the results by a factor which is dependent on the operational environment. A flow chart of this procedure presented in Figure 24.

The short-term energy calculation for a frame *f* of length *N* of speech signal *s*(*n*) is given by Equation (5.1.1). The mean is given by Equation (5.1.3) while the standard deviation is given by Equation (5.1.4).

$$m_{STE} = \frac{1}{19}\sum_{i=1}^{19} STE(i). \tag{5.1.3}$$

$$std_{STE} = \sqrt{\frac{1}{19}\sum_{i=1}^{19}\left(STE(i) - m_{STE}\right)^2}. \tag{5.1.4}$$

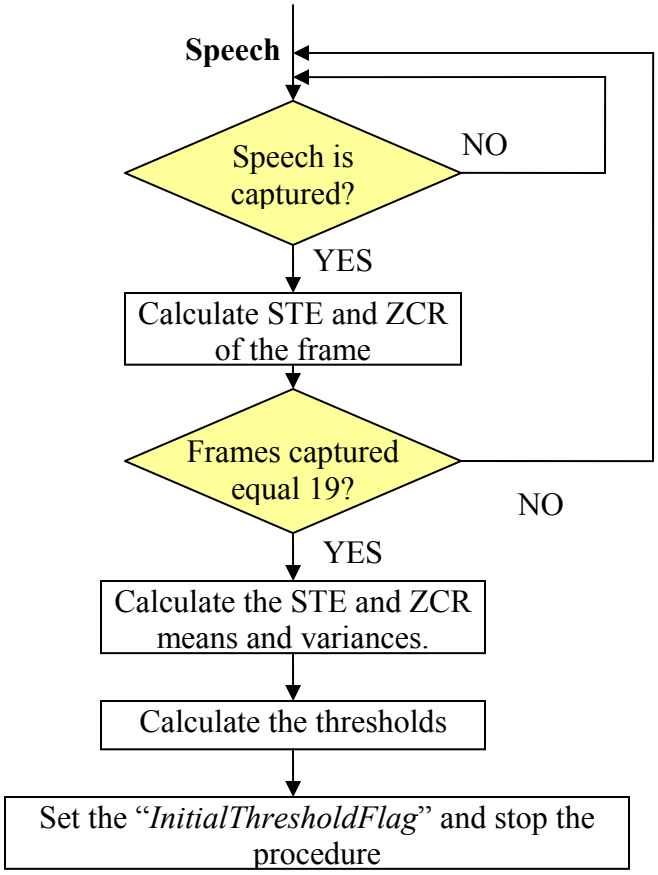

Figure 24. Block Diagram of the Threshold Calculations Procedure

The upper and lower STE thresholds are calculated by multiplying the sum of the mean and the variance by 12 and 3, respectively, as given by Equation (5.1.5), where

these multiplicating factors were derived empirically after trial and error. Finally, an intermediate threshold is calculated between the two thresholds given by Equation (5.1.6).

$$\begin{aligned} STE\_UT &= 12 \cdot \left(m_{STE} + std_{STE} \right), \\ STE\_LT &= 3 \cdot \left(m_{STE} + std_{STE} \right). \end{aligned} \tag{5.1.5}$$

$$STE\_IT = STE\_LT + \left[\frac{1}{8} \left(STE\_UT - STE\_LT \right) \right]. \tag{5.1.6}$$

In previous studies [Kurcan, 2006], an experimental parameter was introduced to Equation (5.1.5) for threshold calculation, to adapt the algorithm to different noise environments. Specifically, instead of using the sum of m_{STE} *and* std_{STE} the minimum value between this parameter and the summation was used. After conducting a series of experiments on the real-time implementation proposed in this study, we concluded that no such parameter was required as the capturing system selected in our study appeared to be more noise robust than that used by [Kurcan, 2006].

The zero-crossing measure (ZCR) calculation is given by Equation (5.1.2). The ZCR mean is given by Equation (5.1.7) and its standard deviation is given by Equation (5.1.8). One threshold calculated for the ZCR quantity, given by Equation (5.1.9), is the minimum value of the summation of mean and standard deviation and the experimental parameter of 20. This parameter is based on experiments on a frame of 80 samples, which demonstrated that 20 zero-crossings may occur just due to noise.

$$mean_{ZCR} = \frac{1}{19} \sum_{i=1}^{19} ZCR(i). \tag{5.1.7}$$

$$std_{ZCR} = \sqrt{\frac{1}{19} \sum_{i=1}^{19} \left(ZCR(i) - m_{ZCR} \right)^2}. \tag{5.1.8}$$

$$ZCR\_T = \min \left\{ 20, \left(mean_{ZCR} + std_{ZCR} \right) \right\}. \tag{5.1.9}$$

The following assembly routines were built to implement the threshold calculations in the microprocessor:

- Threshold_procedure,

- Frame_STE_Calculations,
- Frame_ZCR_Calculations,
- Frame_Threshold_Calculations,
- _Int_SQRT_Calc.

The "*Threshold_procedure*" routine is the general controller of the threshold calculations procedure. It runs every 40 captured samples when and only if the "*InitialThresholdFlag*" semaphore is cleared, meaning that thresholds have not yet been calculated. The routine starts counting the captured speech frames of length equal to 40 samples. When 100 40-sample frames or 4000 samples (500 msec) have been captured, the routine discards all prior samples, keeping only the last 40 samples. From this point and after every additional 40 samples, STE and ZCR values are calculated based on 80-sample frames with overlapping of 50%. STE and ZCR calculations stop after 760 samples or 100 msec and this time is defined as the "silence" period. Next, the routine recalls the "*Frame_Threshold_Calculations*" routine to compute the thresholds. Finally, it sets the "*InitialThresholdFlag*" flag to inform the main program that the thresholds have been calculated and the system is ready for speech processing and recognition, and discards all the samples except the last 40, realigning all the buffers in data memory to their starting points.

"*Frame_STE_Calculations*" and "*Frame_ZCR_Calculations*" routines are implementations of Equations (5.1.1) and (5.1.2) in assembly language. The total amount of time required to run each for a given frame is 660 and 900 pics, respectively. The extracted frame's STE and ZCR values are stored in two different buffers ("FrSTEBuf" and "FrZCRBuf") of length 19 words (corresponding to 19 frames) or 36 bytes each, in data memory.

The "*Frame_Threshold_Calculations*" routine is the implementation of Equations (5.1.3) – (5.1.6) and (5.1.7) – (5.1.9) in assembly language. It takes the STE and ZCR values of the corresponding buffers and the constants parameters (3, 12, and 20) from program memory and computes the thresholds. Finally, it saves the results as global variables in data memory (STE_UT, STE_LT, STE_IT, ZCR_T). This routine runs only once when the "silence" period has been captured and lasts a maximum of 510 pics.

The *"_Int_SQRT_Calc*" routine is an implementation of the integer square root function in assembly language. The algorithm is based on the "Babylonian Approximation" given by Equation (5.1.10). We first use Equation (5.1.10) to obtain the "Babylonian Approximation" of the square root of x. Usually the resultant number is greater than the actual square root of x. Thus, the resultant number is reduced by one, squared and compared to x. The algorithm stops when the first number whose square is less than x is found, resulting in the computation of the floor integer of the square root of x. For example, assume that the square root of the integer x = 3,839 or $(0EFF)_H$ or $(0000\ 1110\ 1111\ 1111)_b$ is required to be calculated. Note that the first '1' present in x from left in the binary representation is in the 12^{th} position, so the approximation given by (5.1.10) is equal to 64. Reducing 64 by one and squaring the result gives 3,969, which is greater than x. The process is continued by repeating the subtraction, squaring the result and comparing it to x. In this case, the procedure is stopped at 61, whose square is equal to 3,721 which is less than x. This is the final result. The routine runs at a maximum of 65 pics.

$$\sqrt{x} \cong 2^{\left(\frac{\text{position of first '1' from left in } x}{2}\right)}. \qquad (5.1.10)$$

3. Normal Procedure

The "*Normal_proc*" procedure executes once the threshold calculations procedure has terminated and while the flag "*InitialThresholdFlag*" is set. This routine calculates the 80-sample frame's STE and ZCR values and controls the start- and end-point detection procedures. Five flags are used for this purpose:

- "FoundTmpStPointFlag": Set means that a possible start-point has been detected and the start-point refinement procedure is ready to run,
- "FinalStartpointFlag": Set means that the start-point refinement procedure has finished and a start-point has been detected,
- "FoundTmpEndpointFlag": Set means a possible end-point has been detected and the end-point refinement procedure is activated,
- "FoundFinalEndPointFlag": Set means that the end-point refinement procedure has run, the captured speech signal's duration is checked and found greater than a low limit and the recognition procedure is ready to yield results,

- "EndPointDetHasRunFlag": Set means that the end-point detection routine has run. This flag is used for the synchronization of the end-point detection and feature extraction routines. This flag is required because the end-point detection routine runs every 40 samples while the feature extraction routine runs every 120 samples.

The normal procedures can be divided into two parts: the start-point detection, including the start-point refinement, and the end-point detection, including the end-point refinement.

a. Start-point Detection Algorithm

The start-point detection algorithm implementation is based on the algorithm described by Kurcan [Kurcan, 2006] and is described by the following steps:

1: Calculate the STE and ZCR for all the frames of the captured signal,

2: Compare the calculated STE values with the Upper STE threshold, beginning at the first frame. At the first occurrence where the STE is greater than the Upper STE threshold, go backward and compare the previous STE values with the intermediate STE threshold. At the first occurrence where the STE value is lower than the intermediate STE threshold, a possible start point has been detected,

3: Go backward seven frames comparing the calculated ZCRs to the ZCR threshold. The frame where the ZCR is found to be greater for third time than the ZCR threshold represents the final start-point. If this point is not found, the start-point is defined as the possible start point detected above.

The main problem with this algorithm is that it cannot be implemented in a real-time fashion. The algorithm requires the entire signal to be captured prior to starting the calculations. This process is not effective in a real-time system for two reasons. First, the captured signal cannot be recorded continuously due to the limited microprocessor's memory capacity. Second, we do not know when the word is spoken. Therefore we do not know when to start recording the captured signal.

Moreover, the use of the upper STE threshold as a trigger for the start-point detection routine is not efficient as it requires processing past frames to estimate the speech start point. The maximum number of frames between the intermediate and upper STE levels was found experimentally to be around 24 40-sample frames or 1000 samples especially for "left" words. These 1000 samples correspond to seven 256-sample overlapping frames. Therefore, the feature extraction and recognition routines must run

seven times before the next 40-sample frame is captured. The time (in pics) required to execute the feature extraction and recognition routines seven times is 206,500 (7 x 29,500 pics per execution, as described in Chapters VI, VIII, and VIII). The available time for their execution is 147,500 pics (the time required for 40 samples to be captured, as described in Section IV.B.2). Thus, the program needs to be able to complete the feature extraction and recognition routines before the next 40-sample frame of speech is captured.

This problem was resolved by designing a new start-point detection algorithm which is described as follows:

1. Calculate STE and ZCR values for each new captured frame,

2. Compare the resultant current frame STE value with the intermediate STE threshold,

3. Set as temporary start-point the current frame, if its STE value is greater than the intermediate STE threshold, and go to step 5,

4. Discard the first 40 samples in the word buffer, realign the frame indices and continue with the next frame (step 1) if step 3 did not give any result for the first seven frames. Hence we store a maximum of seven frames (320 samples), which are needed for the start-point refinement procedure,

5. Continue with step 1 if the current frame's STE is less than the intermediate STE threshold and the number of captured frames is less than 7,

6. Go backward to the stored frames (frame by frame for a maximum of seven frames) comparing the frame ZCR value to the ZCR threshold,

7. Count the number of occurrences where the frame ZCR value is greater than the ZCR threshold,

8. Set as the final start-point the frame where the frame ZCR is found to be greater than the ZCR threshold value for the third time,

9. Set as the final start-point the frame where the temporary start-point was initially determined if step 8 did not give any result.

An accurate detection of the start-point was achieved in real time by implementing this algorithm. The maximum number of possible captured samples stored

in the word's memory data buffer until a start-point is detected is 320 samples. Thus, a maximum of two 256-sample overlap frames may be available (240 for the first frame and 120 more for the second frame, as described in Chapter VI). The time (in pics) required to execute the feature extraction and recognition routines for these two frames is 59,000 (2 x 29,500) pics per execution, as described in Chapter VI). The available time for their execution is 147,500 pics (the time required for 40 samples to be captured, as described in Section IV.B.2). Thus, the program will be able to complete the feature extraction and recognition routines before the next 40-sample frame of speech is captured.

In addition, the routine controls the two flags "*FoundTmpStPointFlag*" and "*FinalStartpointFlag*" used to declare the current status of the algorithm to the normal procedure, as previously described in Section V.A.3.

b. End-point Detection Algorithm

The end-point detection algorithm implementation is based on the algorithm described by Kurcan [Kurcan, 2006] and is described as follows:

1. Calculate the current frame STE and compare it to the Intermediate STE threshold,

2. Set as possible end-point the current frame, if its STE value is lower than the intermediate STE threshold,

3. Continue with step 1 if step 2 did not give any result,

4. Continue to capture frames, calculating the current frame's ZCR,

5. Compare the current frames's ZCR to the ZCR threshold (seeking for the former to be greater than the latter),

6. Set as final end-point the frame where for the third time the ZCR is found to be greater than the ZCR threshold, and continue with step 8,

7. Set as final end-point the frame where the possible end-point was determined if the step five did not give any result for seven consecutive frames after step 2,

8. Calculate the duration of the captured word as the total number of the stored word's samples,

9. Compare the word's duration (calculated in step 8) with the minimum acceptable number of samples that may contain a captured word (in this study the number is 1000 samples),

10. Continue with the word recognition routine if the captured word's duration is greater than 1000 samples,

11. Delete all captured samples in the word buffer (except the last 40), realign buffers and indices, clear all flags associated with the end-point detection routines. Start again from the start-point detection routine, if the duration is less than 1000 samples, as in this case noise or "spikes" have been captured instead of speech.

By implementing this algorithm, an accurate detection of the end-point is achieved in real time. In addition, the final recognition routine can be run immediately after the end-point detection, providing recognition results before the next 40-sample frame is captured, as described in Chapter IIX. Samples collected after the end-point are ignored. Furthermore, the algorithm controls the two flags used from the normal procedure "FoundTmpEndpointFlag" and "FoundFinalEndPointFlag", declaring the current status of the algorithm.

c. End-point Detection Algorithm Results

Figure 25 shows a portion of speech, in the time domain, containing the word "left." It is captured by the audio codec and processed by the microprocessor. Figure 26 shows the STE plot of the same signal for each 80-sample, 50% overlapped frame, and the STE thresholds calculated by the thresholds calculation routine. Note that the STE value in the "silence" period is below the low STE threshold, before and after the word. In this example, the application of the end-point detection algorithm results in two end-points denoted by lines N1 and N2, depicted in Figure 26. These points represent the temporary start and end point of the word "left." In this same plot, the lines at N1' and N2' correspond to the final start and end points of the word "left" resulting from the refinement routine used in conjunction with the ZCR measure. The ZCR measure helps the end-point detection algorithm detect the weak fricative "t" at the end of the word "left" and the "l" before the high energy vowel "e" at the beginning of the word. As shown in Figure 27, these points are obtained by the ZCR measure and are identified where the frame ZCR is greater than the ZCR threshold for more than three times in 7-frames regions outside the word's STE limits.

Figure 25 also shows the final bounds of the word “left.” Note that the algorithm does not capture the whole word, demonstrating that it is not always capable of finding the exact bounds of a given word, even using the ZCR measure refinement. Finally, Figure 28 shows the cropped word “left” which is used for features extraction and recognition.

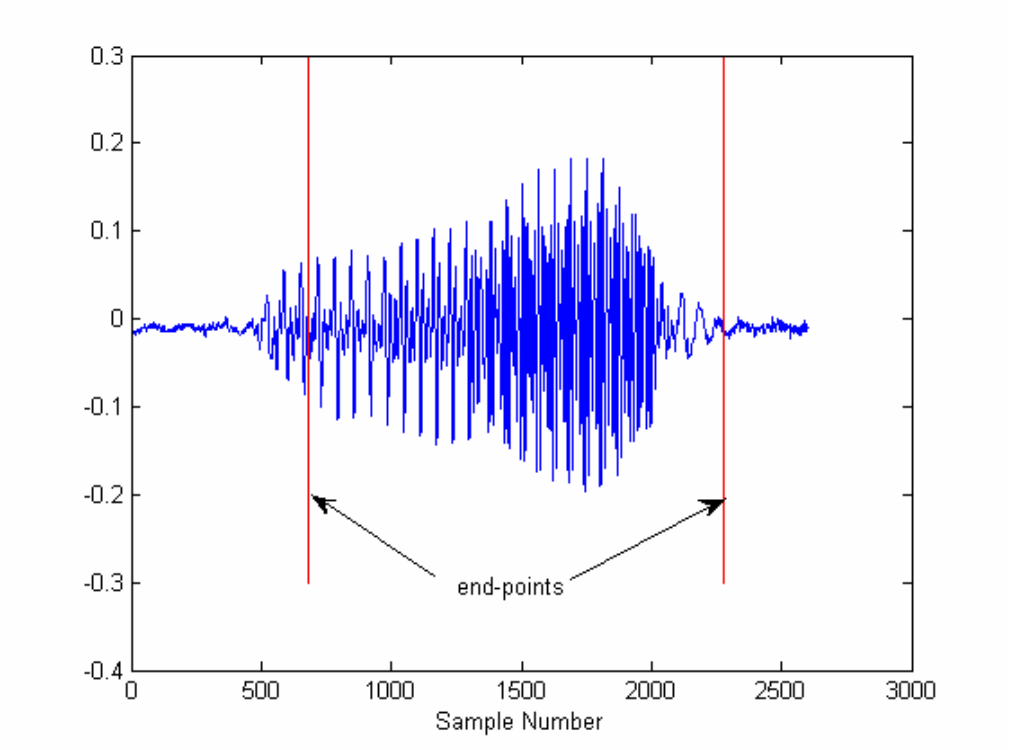

Figure 25. Captured Speech Signal Containing the Word “left”

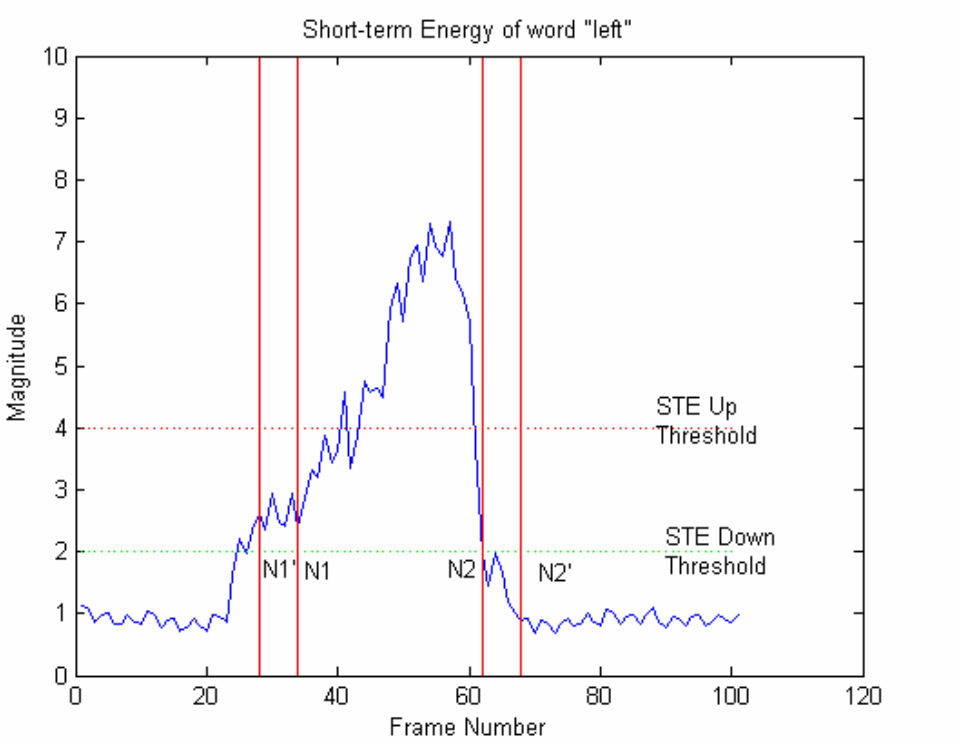

Figure 26. Short-term Energy Plot of Captured Signal Containing the Word “left”

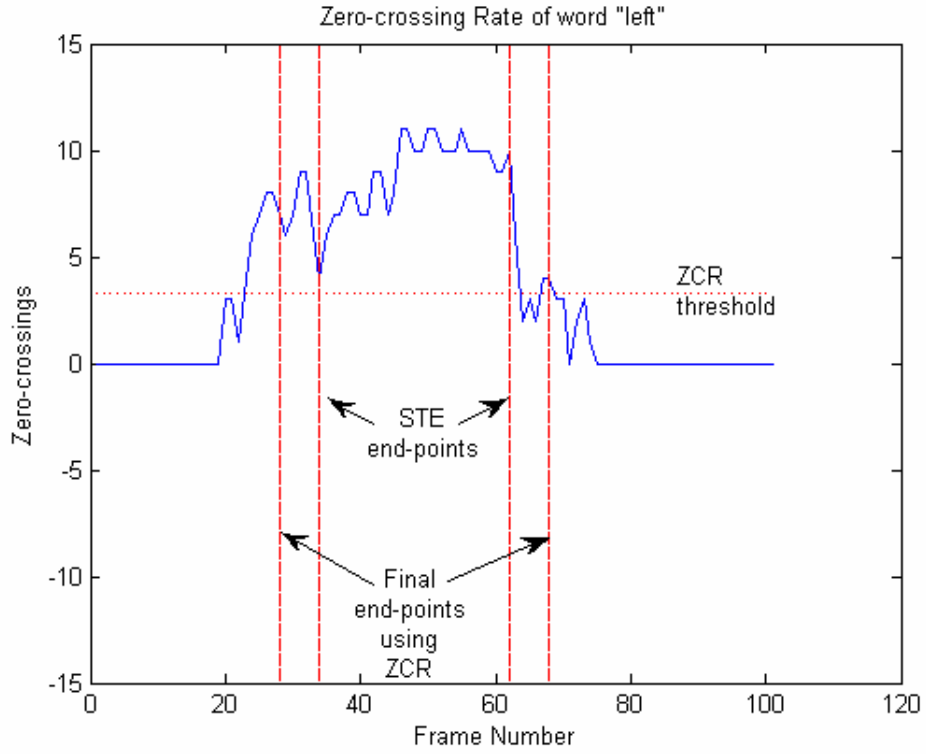

Figure 27. Zero-crossing Rate Plot of Captured Signal Containing the Word “left”

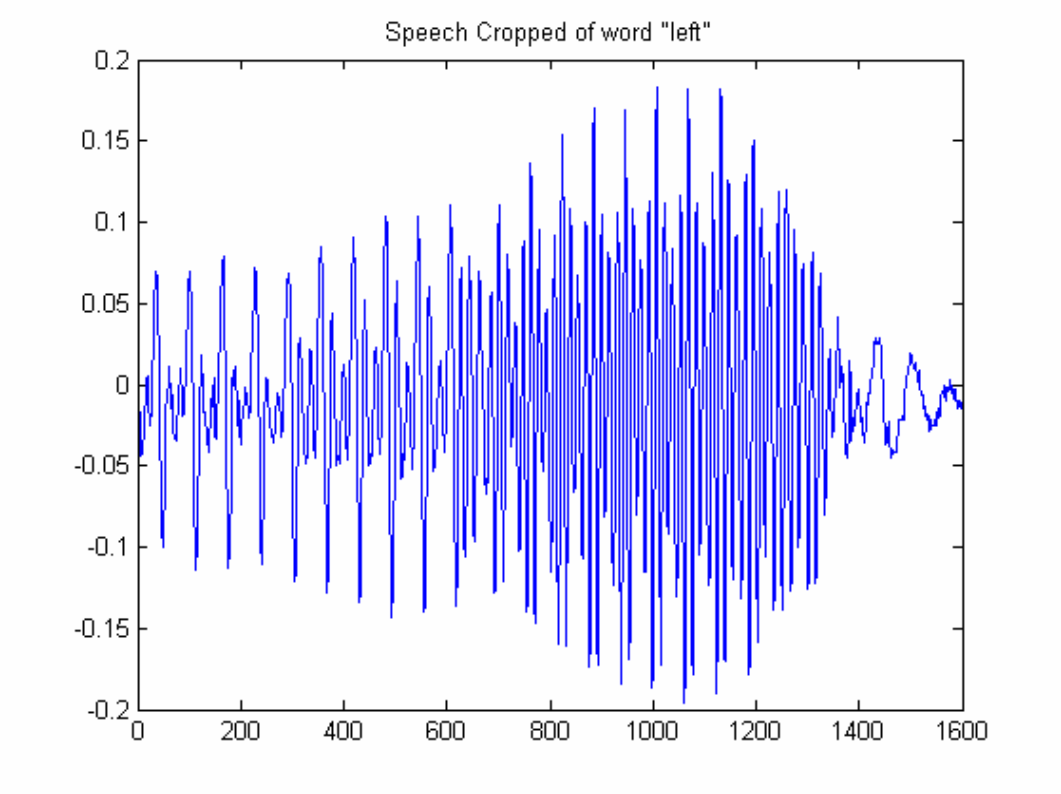

Figure 28. Final Cropped Speech Signal of Word “left”

The detected end-points of the spoken word are used as trigger for the activation and termination of the feature extraction algorithm. Next, we discuss the feature extraction algorithm implemented in this study to achieve a real-time speech processing and recognition system.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. SPEECH FEATURE EXTRACTION

A. INTRODUCTION

Speech recognition has better performance when the recognizer is fed with compact feature vectors. As discussed in Chapter II, Mel-Frequency Cepstral Coefficients (MFCCs) are widely accepted and used to represent speech signals, preserving the speech characteristics, while reducing the effects of speech variability [Deller 2000]. Furthermore, Davis et al. concluded that MFCC features outperform other types of speech signal representation, especially when used for monosyllabic word recognition [Davis, 1980]. Moreover, Kurcan showed that MFCC features yielded better results than other parameters considered and were effective in the isolated word recognition case [Kurcan, 2006], which is the focus of this study. The same approach is implemented in this thesis for ear-microphone speech signal feature representation.

Figure 29 shows a block diagram of a theoretical MFCC feature extraction process and is described as follows [Kurcan, 2006]:

- Framing: Speech data are framed in 256-sample frames corresponding to 32 msec, overlapped by 53%, to better capture temporal changes from frame to frame.
- Windowing: Speech frames are windowed by applying a Hamming window $w(n)$, given as

$$w(n) = \begin{cases} 0.54 - 0.46\cos\left(\dfrac{2\pi(n-1)}{N}\right), & n = 1,\ldots,N = 256 \\ 0, & \text{otherwise.} \end{cases} \tag{6.1.1}$$

- Fast Fourier Transform (FFT): In each frame, a complex 256-Fast Fourier Transform is applied to transform the signal from the time to the frequency domain.
- Mel-frequency Warping: The frequency information obtained in each speech frame is passed through the Mel filter-bank described earlier, resulting in 24 frequency coefficients per frame.
- Log Energy Computation: A logarithm transformation is applied to the magnitude of each mel frequency coefficients, discarding the phase information, dynamically compressing the features, and making feature extraction less sensitive to speaker-dependent variations [Becchetti, 1999].

Mel-frequency Cepstrum Computation: Finally, the mel-frequency cepstral coefficients are computed by applying the inverse DFT to the logarithm of the magnitude of the filter-bank outputs. Note that the inverse DFT reduces to a Discrete Cosine Transform (DCT) operation as the log magnitude spectra of the coefficients are real and symmetric [Becchetti, 1999]. Moreover, the DCT has the advantage of producing highly uncorrelated features [Jayant, 1984; Deng 2003]. The resulting Mel-frequency cepstral coefficients $c(k)$ are given by

$$c(k) = w(k)\sum_{n=1}^{M} \log\left(\left|x_k(n)\right|\right)\cos\left(\frac{\pi(2n-1)(k-1)}{2M}\right),\ k = 1,...,L, \qquad (6.1.2)$$

where

$$w(k) = \begin{cases} \dfrac{1}{\sqrt{M}}, & k = 1, \\ \sqrt{\dfrac{2}{M}}, & 2 \le k \le L, \end{cases} \qquad (6.1.3)$$

$x_k(k)$ are the 24 frequency coefficients resulting from the mel filter-banks, M=24, and L=14 equal to the number of MFCC coefficients selected per speech frame.

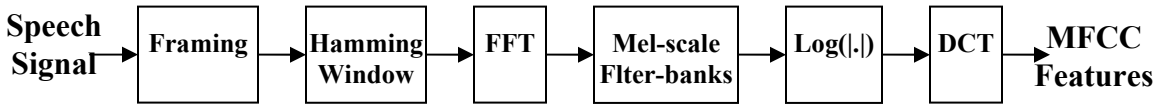

Figure 29. Theoretical MFCC Feature Extraction Block Diagram

Next, we describe the real-time implementation of the feature extraction algorithm.

B. FEATURE EXTRACTION REAL-TIME IMPLEMENTATION

1. Feature Extraction Procedure General Description

The feature extraction algorithm described in Section II.B.3, includes a series of computational steps, such as the FFT, DCT, and mel filter-banks calculation, which demand a great number of calculations. In addition, a real-time ASR system must

perform all these operations on a frame by frame basis, in an amount of time less than that required to capture a frame's speech samples. Furthermore, the feature extraction algorithm in a real time implementation must run in parallel with the end-point detection algorithm because the first routine stops when the second one detects a final end-point. Synchronization and timing problems may occur due to the different execution rate of these two algorithms. Specifically, a 256-sample overlap frame, or 120 additional captured samples after those associated to the last frame, must be available for the activation of the feature extraction algorithm (as described in the next paragraph) while the end-point detection algorithm is activated every 40 new captured samples (as described in Section V.A.3.b) Thus, a mechanism is required to synchronize these two algorithms. As a result, the feature extraction algorithm previously described was redesigned to meet the system requirements necessary for the real-time implementation considered in this thesis.

The redesign of the feature extraction algorithm was based on two factors; first the choice of the amount of the overlap in the 256-sample frame, second the solution of the synchronization and timing problems previous described.

The overlap amount of the 256-sample frame was chosen based mainly on the number of the new samples required to form the next frame. Note that the number of new samples must be a multiple of the theoretical timer of 40-sample frames. Thus, an overlap of 53% was chosen as the 256-sample frame is formed by the last 136 samples of the previous frame and 120 additional new captured samples. Consequently, the 256-sample frame will be available when every new three 40-sample frames (120 samples) are captured. Therefore, the algorithm will be synchronized with the 40-sample theoretical timer, as described in Section IV.B.2.

The algorithm must start immediately after a final start-point has been detected and must stop immediately after a final end-point has been detected. Due to the difference in the activation timer of the end-point routines and the feature extraction routine (every 40 samples for the former while every 120 samples for the later) the end-point may be detected before or after the end of the 256-sample frame. Thus, the

redesigned algorithm must include a mechanism to synchronize the actual detected end-points with the first and the last 256-sample frames used for feature extraction.

Taking into account the previous described factors we redesigned the algorithm by separating it in two procedures:

- Feature extraction activation procedure (*Feature_extraction_activation*): Responsible for synchronization of end-points routines with the feature extraction procedure,
- Feature extraction procedure (*Feature_extraction_proc*): Responsible for activating the routines (computational steps) required to calculate the MFCC parameters.

Furthermore, two extra word buffer's pointers, and two flags needed to be introduced in the main program:

- FrameSpInputSignalBufPtr: Points to the 256-sample frame's end-address in the word's buffer,
- FEPFrameStartPoint: Points to the previous 256-sample frame's end-address in the word's buffer,
- FirstTimeFEPFlag: Indicates that the feature extraction algorithm is going to run for the first time,
- EndPointDetHasRunFlag: Indicates that the end-point detection has been executed.

The exact role of each pointer and flag in the program is described in the following two sections.

2. Feature Extraction Activation Procedure

The feature extraction activation procedure (FEAP) controls the real-time execution of the feature extraction algorithm on a 256-sample frame basis. The procedure is activated immediately after a start-point has been finally detected. Then, the possible word's data buffer's length values are multiples of 40 with a maximum of 320 and a minimum of 80 samples. Note that two problems were introduced at this point because the number of samples needed to form the first frame (ready for feature extraction) is equal to 256; first 256 is not divisible by 40, and second the number of captured samples may be above or below 256 (because the start-point detection procedure may finally detect the start-point between the first and the 7^{th} 40-sample frame, as described in Section V.A.3.a).

The first problem is a synchronization problem. Every routine in the program must be synchronized with the theoretical timer of 40 samples, as described in Section IV.B.2. On the other hand, the first frame uses 256 samples (no overlap). If these 256 samples were used to form the first frame then the next 256-sample overlap frame will be completed after an additional 120 samples are collected, i.e., 376 samples. Note this number is also not divisible by 40. Thus, the program (running based on the 40 samples theoretical timer) is not able to synchronize the frame completion with all other routines running in the same amount of time (end-point detection, DCI interrupt). This problem was solved by forming the first frame using the first 240 captured samples (this number is the closest to 256 which is divisible by 40), and padding 16 zeros at the beginning of the frame. Consequently, the next 256-sample overlap frame will be available after an additional 120 samples are collected or 360 total samples, which is also divisible by 40. Thus, the program now is able to synchronize the completion of the 256-sample frame with the theoretical timer of 40 samples.

The second problem was solved by using the 240 samples previously described, as criterion for program data flow. When a start-point is finally detected, the start-point procedure deletes all the samples in the word's data buffer until that point. Therefore, 40 to 320 samples may exist in the buffer, depended on where the start-point was detected (Section V.A.3a). The total number of samples remaining in the buffer is compared with 240. If this number is below 240 (i.e., 80, 120, 160, or 200) the feature extraction procedure (FEAP) is not able to form the first frame (240 sample). Thus, it continues to capture samples until they reach 240. Then the FEAP forms the first frame by padding 16 zeros at the beginning of the frame and activates the feature extraction procedure. The next frame (second one) will be ready after 120 additional samples are captured. Then, and after every 120 new samples are collected, the FEAP forms the 256-sample frame using the last 256 samples in the word's data buffer (136 samples from the previous frame and 120 new captured samples). On the other hand, if the number of samples in the word's data buffer is greater than 240 (280, or 320) when the start-point is detected, the FEAP uses again the first 240 to form the first frame (padding again with 16 zeros at the beginning) and activates the feature extraction procedure. The next frame (second one) will be ready after 80 or 40 new samples have been captured, because 40 (280-240) or 80

(320-240) samples of the second frame are already stored in the word's data buffer. Then, after every 120 new samples are collected the FEAP forms the 256-sample frame using the last 256 samples in the word's data buffer.

The FEAP stops to activate the feature extraction procedure when a final end-point is detected. A data conflict may be introduced at this point mainly due to the different activation timing of the end-point detection routine and the feature extraction procedure (every 40 samples for the former while every 120 samples for the later). According to the end-point detection routine (including the end-point refinement routine) described in Section V.A.3b, first a possible end-point is detected. Next, the program continues to capture samples for seven more 40-sample frames (corresponding to 320 samples; 7x40=280 plus 40 of the first frame due to the overlap). The end-point refinement routine seeks the final end-point between these frames comparing the ZCR quantity. Thus, two different cases are possible, as described in Section V.A.3b; the end-point is finally detected between the 3rd and the 7th 40-samples frame, or the end-point is finally the possible end-point previously detected. In the first case, all captured samples until the final end-point (3rd, 4th, 5th, 6th or 7th 40-sample frame after the possible end-point) are stored in the word's data buffer. In the second case, all captured samples until the 7th (after the possible end-point) 40-sample frame are stored in the word's data buffer because the refinement routine has run the seven frames without finding a final end-point. Moreover, the features may have been extracted for all or for a part of these samples, because the FEAP stops to activate the feature extraction procedure only when an end-point is finally determined. Thus, we had to design a mechanism which synchronizes the final end-point with the end of the 256-sample overlap frame used for feature extraction to avoid potential data conflicts. This mechanism is based on two pointers introduced in Section V.A.3; the *FEPFrameStartPoint* pointer which is used here to indicate the address of the end of the last 256-sample frame used for feature extraction, in the word's data buffer, and the *SpInputSignalBufPtr* pointer which is used here to indicate the address of the end of the 40-sample frame where the end-point was detected. FEAP checks the difference between these two pointers (*SpInputSignalBufPtr* – *FEPFrameStartPoint*) resulting in six different cases:

1. Equal to 0: This means that the features are available for all captured samples and FEAP ends,

2. Equal to 40: This means that the features for the last 40 samples have not been calculated. FEAP forms the last 256-sample frame by padding the frame end with 80 zeros, and activates the feature extraction procedure for the last time. After this the FEAP ends,

3. Equal to 80: This means that the features for the last 80 samples have not been calculated. FEAP forms the last 256-sample frame by padding at the end 40 zeros, and activates the feature extraction procedure for the last time. After this the FEAP ends,

4. Equal to -200: This means that features have been calculated for 200 samples after the end-point. Then the FEAP deletes the last two sets of features corresponding to the last 240 samples (as FEAP activates the feature extraction process every 120 samples). The end-point is moved backward 40 samples to be synchronized with the features (200-240=-40). After this the FEAP ends,

5. Equal to -240: This means that the features have been calculated for 240 samples after the end-point. Then the FEAP deletes the last two sets of features corresponding to the last 240 samples (as FEAP activates the feature extraction every 120 samples). The end-point is not affected because it is synchronized with the features (240-240=0). After this the FEAP ends,

6. Equal to -280: This means that the features have been calculated for 280 samples after the end-point. Then the FEAP deletes the last two sets of features corresponding to the last 240 samples (as FEAP activates feature extraction every 120 samples). The end-point is moving forward for 40 samples to be synchronized with the features (280-240=40). After this the FEAP ends.

Cases 1 to 3 above refer to the event where the final end-point is detected between the 3rd and 7th (last frame) 40-sample frame past the temporary end-point. Cases 4 to 6 refer to the event where the final end-point is detected at the temporary end-point meaning that the refinement procedure has run for more than seven 40-sample frames without a result.

Figures 30 and 31 show two different examples of the FEAP frame synchronization approaches described above. In Figure 30, the final end-point is detected six frames after the temporary end-point location. The difference between the *SpInputSignalBufPtr* and *FEPFrameStartPoint* pointers is 80 samples. Thus, FEAP

forms the last 256-sample overlap frame padding 80 zeros at the end, and activates the feature extraction procedure for one more time.

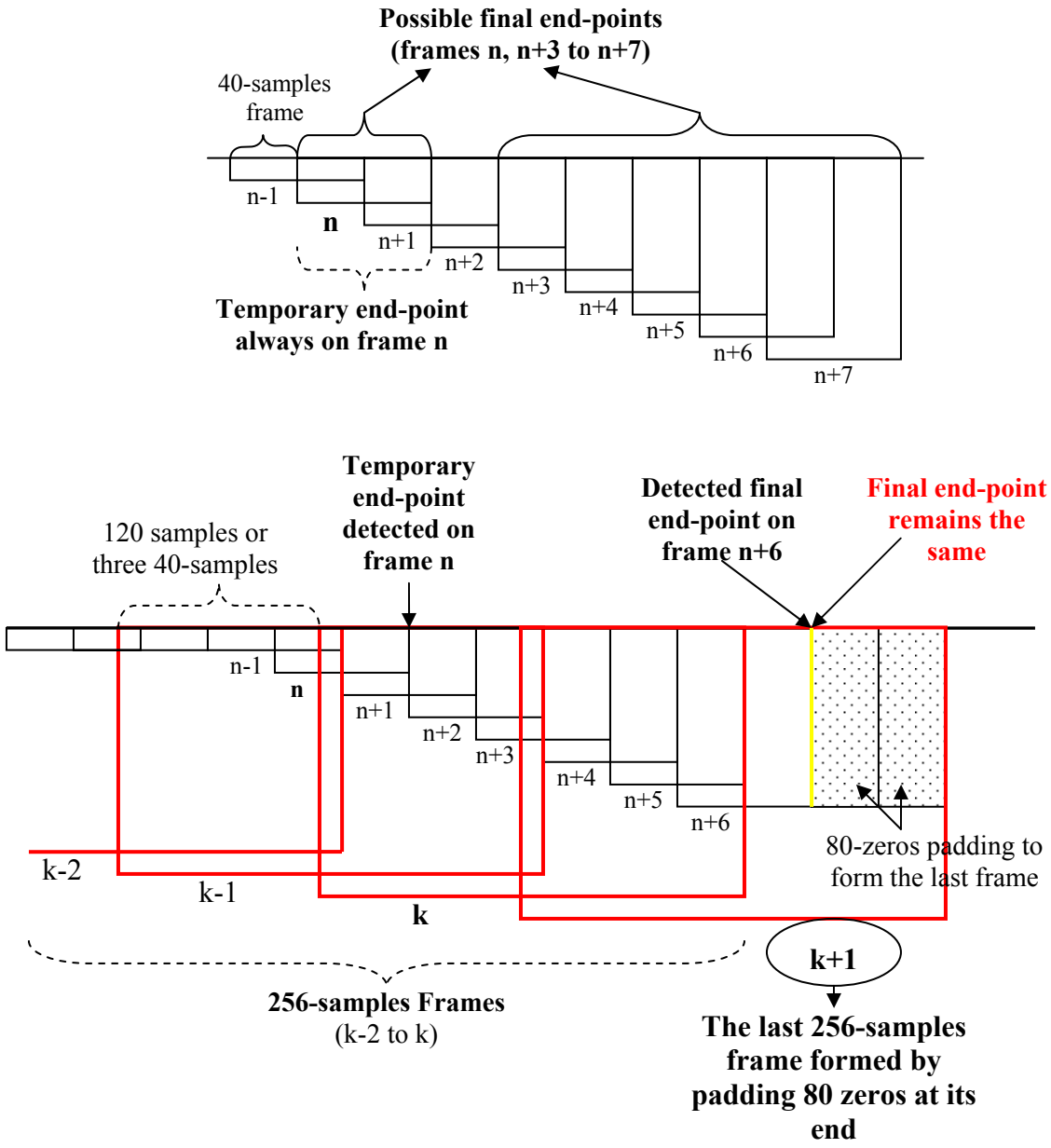

Figure 30. Feature Extraction Activation Procedure, Example 1

In Figure 31, the final end-point is detected at the temporary end-point. The difference between the *SpInputSignalBufPtr* and *FEPFrameStartPoint* pointers is -200 samples. Thus, the final end-point is moved 40 samples backward and the last two sets of calculated features are deleted.

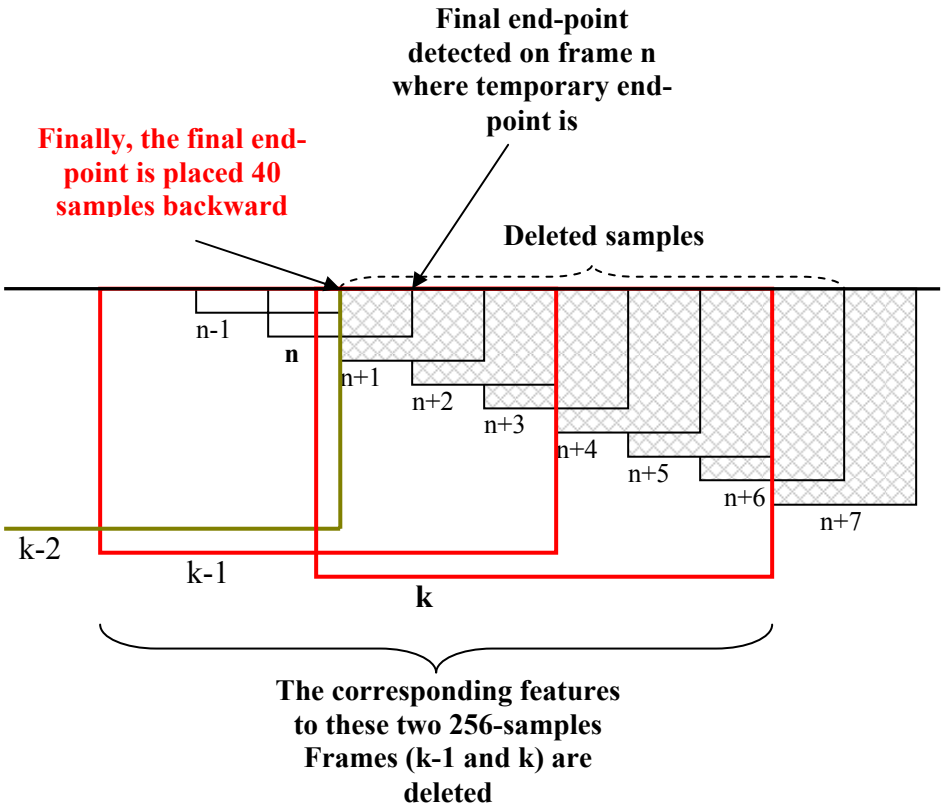

Figure 31. Feature Extraction Activation Procedure, Example 2

FEAP must run immediately after the final start-point is detected. The combination of these two routines can exceed the 4-sample theoretical timer of the DCI interrupt (14,750 pics) introducing data conflicts. This problem was solved by using the *EndPointDetHasRunFlag* flag and the *FrameSpInputSignalBufPtr* word's buffer pointer. The flag is set by the end-point detection routine every time it is activated and is cleared when FEAP finishes. The FEAP procedure is activated only if this flag is set. Therefore, the FEAP runs only if the end-point detection routine has been executed. The *FrameSpInputSignalBufPtr* word's buffer pointer is used to declare the 256-samples

current frame's end-address in the word's buffer. Thus, the frame's boundaries are kept constant until the next 40-sample frame is captured. Then the pointer is updated by the DCI interrupt. Experimentally we showed that the feature extraction step (including the end-point detection) is completed after two DCI interrupts or eight new samples, consuming approximately 22,400 pics.

The actual feature extraction routines, activated by the FEAP, are included in the Feature Extraction Procedure described next.

3. Feature Extraction Procedure

The feature extraction procedure (FEP) is one of the most resource-consuming routines in this system. For each 256-sample overlap frame, a hamming window, a complex FFT, magnitude and logarithm calculations, mel Filter-Bank filtering, and finally a discrete cosine transformation are executed. FEP is responsible for initializing the indices, the buffers and their pointers and to recall, one by one, all the routines corresponding to the above calculations.

The feature extraction calculations are implemented based on the dsPIC33FJ256GP710 microprocessor's Digital Signal Processing (DSP) core commands and architecture described in Chapter IV. Further, the data memory is split into two different regions "x" and "y" to achieve less computation cycles. Five data buffers are used, placed either in "x" or "y" regions of data memory as follows:

- *ANxInputSignalBuf*: A buffer in "x" data memory of 512 words length. Used to store the 256-sample frame at the beginning of the FEP procedure. Also used as input and output buffer in the hamming window routine and the FFT routine and as input to the magnitude routine,
- MagnitudeBuf: A buffer in "x" data memory of 256 words length. Used to store the magnitude of the 256 frequency bins resulting from the FFT routine. Used as output by the magnitude routine and as input to the mel filter-bank filtering routine,
- TempMelBuf: A buffer in "x" data memory of 24 words length. Used to temporarily store the 24 frequency bins resulting from the mel filter-bank filtering routine. Also used as an input and output to the logarithm calculation routine and, consequently, the DCT routine,
- TmpDCTBuf: A buffer in "y" data memory of 14 words length. Used to temporarily store the 14 MFCC parameters extracted by the DCT routine. Also used as an input to the codebook routine described in Chapter VII,

- y_twid: A buffer in "y" data memory of 256 words length. Used to store the FFT coefficients required for the fast Fourier transformation.

The following assembly routines were built for the feature extraction procedure microprocessor implementation:

- _hamming_proc,
- _fft_proc,
- _magnitude_proc,
- _melbank_proc,
- _log2magn_proc,
- _dct_proc.

The "*_hamming_proc*" routine is an assembly language implementation of Equation (5.1.1). The 256-sample frame stored in "*ANxInputSignalBuf*" is multiplied in the time domain with hamming window coefficients that are pre-calculated and hard-coded in program memory. The results are stored in the same buffer. The total amount of time necessary to run for each frame is 792 pics.

The "*_fft_proc*" routine is an implementation of the fast Fourier transformation using the radix-2 "butterfly" technique [Cristi, 2004]. For a data frame of length $N=2^L$, the radix-2 FFT's complexity increases as order $O\{N \log 2N\}$. In contrast, the discrete Fourier transformation's (DFT) complexity increases much faster, as $O\{N^2\}$. In addition, using the microprocessor's DSP commands (where an addition and a multiplication can be executed in one cycle and in the same cycle the next samples can be loaded) the FFT transformation can be completed in less than 7,000 pics. Note that the real samples are converted into complex format by padding zeroes appropriately in the FFT buffer "ANxInputSignalBuf" prior to the complex FFT implementation. Furthermore, the input samples are reordered in a binary reversed order as required by the "butterfly" calculation scheme [Cristi, 2004]. The 256 2-word complex numbers (frequencies) are stored in the same buffer in normal order. The 256 pre-calculated FFT coefficients are loaded in "y" data memory. Figure 32 shows the "butterfly" operation scheme, while Figure 33 shows an example of a 4-point DFT, implemented using the radix-2 FFT.

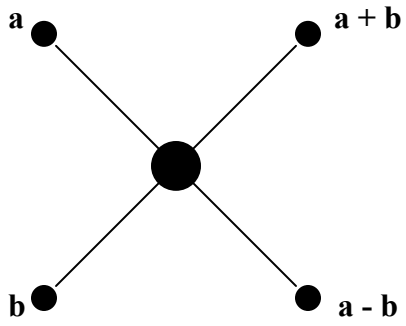

Figure 32. Butterfly Operator

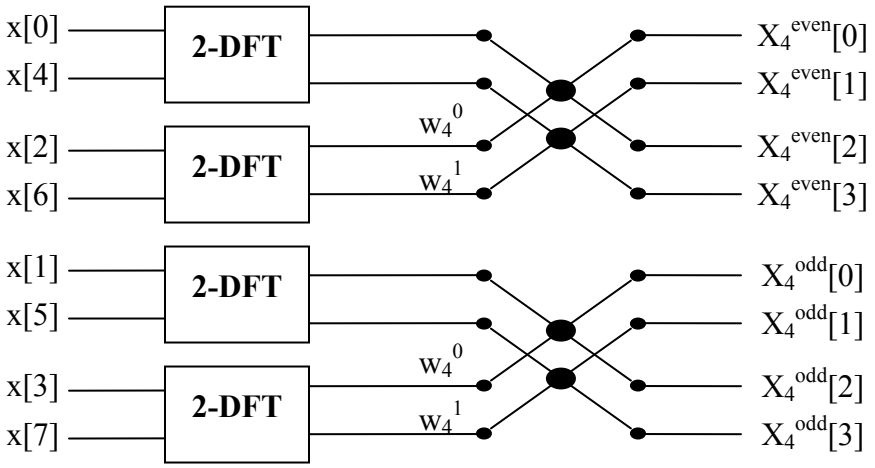

Figure 33. 4-point DFT Implementation Using Radix-2 FFT

The "*_magnitude_proc*" routine is an assembly language implementation of the calculation of the absolute value of the frame's frequency bins resulting from the FFT routine, based on the following equation:

$$\left| y_{complex} \right| = y_{real} \cdot conj\left(y_{imag} \right). \tag{6.3.1}$$

Note that only the magnitudes of the first 128 values are computed as the FFT's resultant frequency bins are symmetric about the origin. The resultant 128 words representing the magnitude are stored in "*MagnitudeBuf*." The total amount of time required for the routine execution is 780 pics.

The "*_melbank_proc*" routine is an assembly language implementation of the mel bank-filters filtering algorithm. The filtering operation is accomplished in the frequency domain, resulting in a series of multiplications between the 128 frame's frequency bins and the filter-bank filters' coefficients. The coefficients of 24 triangular overlapped filters are calculated, converted in hexadecimal (1.15) format, and hard-coded in program memory during the microprocessor initialization. Normally, 425,900 multiplications and 16,384 additions are required to pass 128 frequency bins through 24 different overlapped filters, which exceed the microprocessor's specifications for a real-time system. Thus, the algorithm was designed based on the fact that most of the filters' coefficients are equal to zero. The idea is to take each filter separately and multiply its non-zero coefficients with the corresponding frame's frequency bins. From each filter, we finally obtain one frequency bin (the magnitude only). This is calculated for each of the 24 filters by the summation of the filters' multiplication's results above. Finally, 24 mel frequency bins are calculated and stored in "*TempMelBuf"* in data memory. Using this algorithm mel bank-filter filtering is accomplished in 7,300 pics.

The "*_log2magn_proc*" routine is an assembly language implementation of the calculation of the logarithm of the mel filter-bank frequency magnitudes. A base-2 logarithm calculation is used, based on the following formula [Parhami, 2000]:

$$\log_2(x) = \log_2(2^q s) = q + \log_2(s), \quad \text{where } 1 \le s < 2. \tag{6.3.2}$$

Magnitudes are assumed to be integers and a binary representation is used. Thus, the number q is the floor of the $log_2(x)$ given by the following form:

$$floor\left[\log_2\left(x_{16\text{-binary}}\right)\right] = \begin{cases} q = 16 - n & , n > 0, \\ q = 0 & , n = 0, \end{cases} \tag{6.3.3}$$

where n is the position of the first "1" from the left of the number x expressed in a binary format.

The value s is given in Equation (6.3.4) and belongs to the interval [1,2). The logarithm y in base two of s is given by a lookup table. We implemented a DELPHI program to calculate the lookup table and converted it in hexadecimal 1.15 fractional format. The range of s numbers starts from 1.0005 and stops at 1.9995 in 0.0005

increasing step. The logarithms y belong to the interval (0, 1) and a total of 1,999 numbers are computed. The resultant logarithm's table is hard-coded in the microprocessor's program memory requiring 3,998 bytes.

$$s = \frac{x}{2^q} \tag{6.3.4}$$

Using the above equations the logarithm calculation of an integer number x can be summarized by the following steps:

1. Find q using Equation (6.3.3),
2. Calculate s using Equation (6.3.4). The fractional part of the result plus one is equal to the number s,
3. Find in the lookup table the corresponding logarithm y to the number s,
4. Logical shift q left by 11 positions,
5. Logical shift right the logarithm of s by four positions,
6. Add the two binary numbers.

Up to step 3, q is a binary representation of an integer number while y is a binary representation in 1.15 format of a fractional number. In addition the final result of the logarithm of x must be in 1.15 fractional format. Next, we divide both numbers q and y by 16 so that they can be preserved into the final result. The division by 16 of q leads to a fractional number in the interval [0.0625, 0.9375] (from Equation (5.3.3) q is an integer less than 16 and greater than 0). This division in binary arithmetic can be done by shifting q left by 11 positions (step 4). The division by 16 of y (which is already in 1.15 format) can be done in binary arithmetic by shifting y to the right by four positions (step 5). Now we can add q and y to take the final approximation of the logarithm of x divided by 16. The total amount of time to run for each frame is 3,300 pics.

The "*_dct_proc*" routine is an assembly language implementation of Equation (5.1.2). The 24 logarithms stored in the "*TempMelBuf*" buffer are transformed into 14 cepstral coefficients using a discrete cosine transformation. The results are stored in the "*DCTTempBuf*". The calculations are performed in 1.15 fractional format. Note that DCT coefficients are pre-calculated and hard-coded into program memory during the microprocessor's programming. The total amount of time required to run for each frame is 1,100 pics.

4. Feature Extraction Results

This thesis designed a real-time feature extraction procedure, synchronized with the end-point detection routine, using up to 20% of the available processor's time before the next 40-sample frame is presented. The performance is achieved by optimizing the calculations and taking advantage of the microprocessor's DSP core architecture and commands. Fourteen MFCC coefficients are extracted for each 256-sample frame.

Figure 34 and 35 shows the FFT frequency bins of the first frame of the words "left" and "down", respectively. The energy, as expected due to the ear microphone, is concentrated at the low frequencies. Figure 36 and 37 show the extracted MFFC parameters of words "left" and "down" respectively.

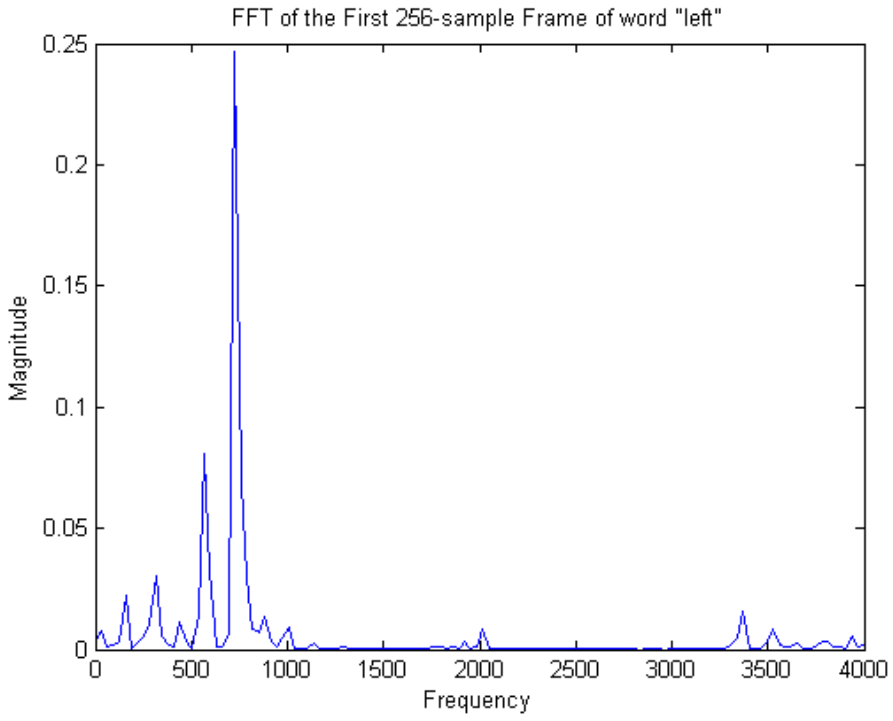

Figure 34. Fast Fourier Transform of the First 256-sample Frame of Word "left"

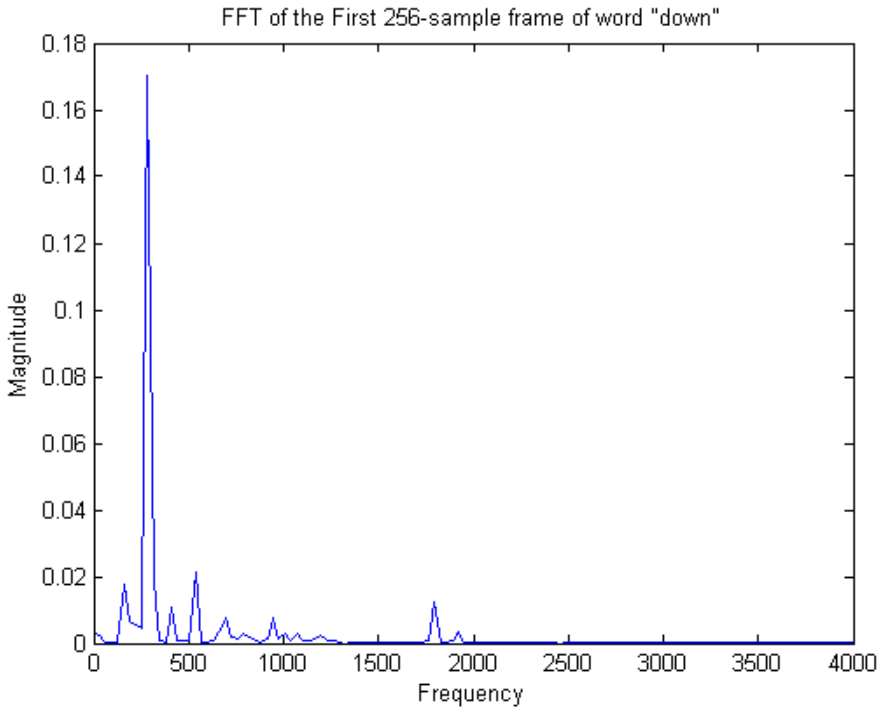

Figure 35. Fast Fourier Transform of the First 256-sample Frame of Word “down”

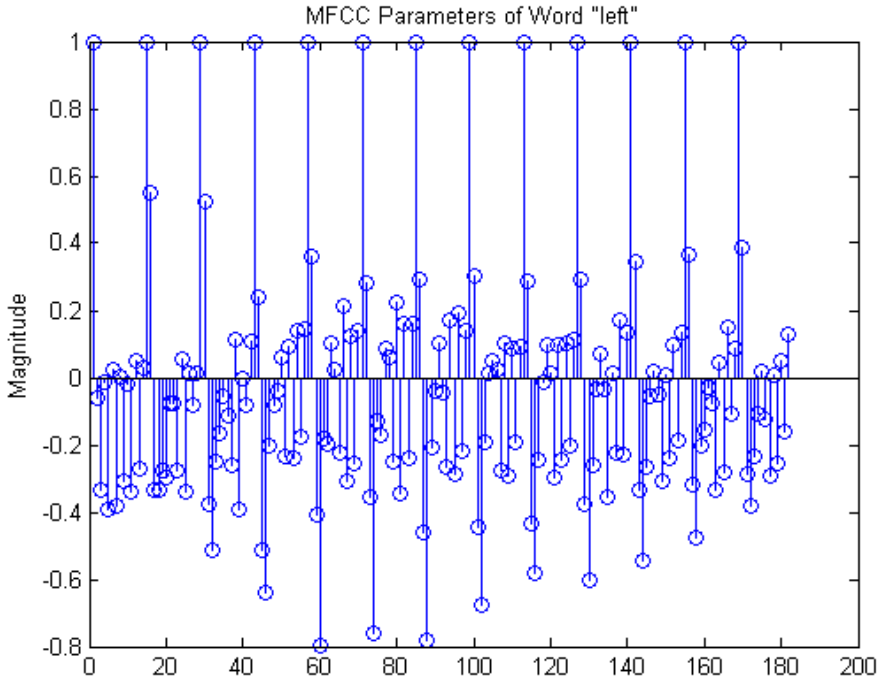

Figure 36. MFCC Parameters of Word “left”

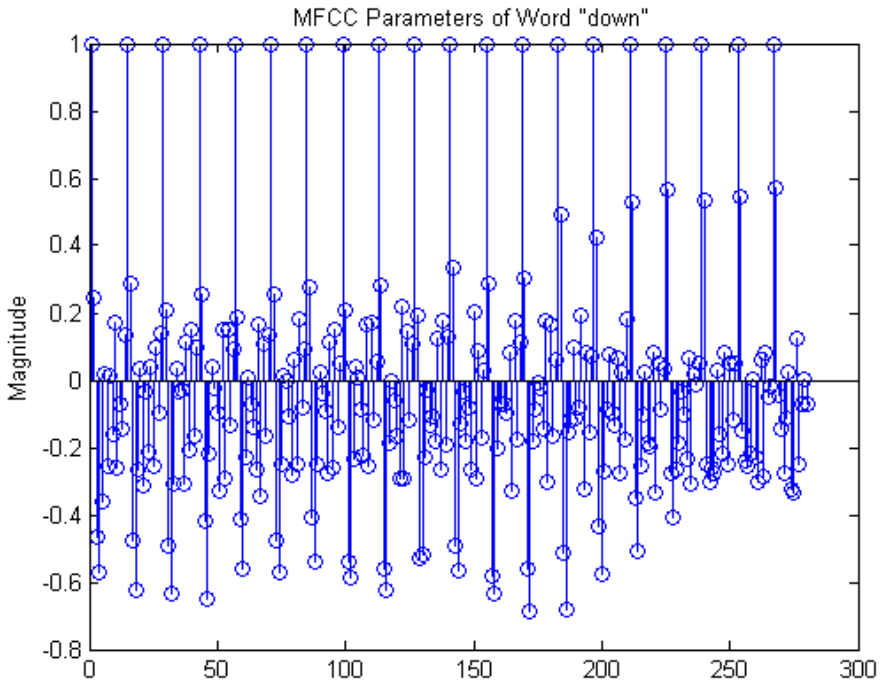

Figure 37. MFCC Parameters of Word "down"

Twelve of the fourteen (2nd to 13th coefficient) MFFC parameters extracted for the current 256-sample frame are used in the recognition task. The next step is to represent these 12 parameters by a discrete code number. Thus, the features are quantized by a 12x128 vector quantizer. Next, we analytically describe the implementation of the MFCC parameters' coding using a 12x128 vector quantizer.

THIS PAGE INTENTIONALLY LEFT BLANK

VII. SPEECH FEATURE QUANTIZATION

A. INTRODUCTION

Vector quantization schemes convert the continuous amplitude MFCC parameters into discrete symbols (codes), removing any possible redundancy. These vector codes correspond to different clusters in regions near a center (centroid). A code associated with each cluster is used to represent a frame's features vector in the DHMM recognizer.

The K-means iterative algorithm implemented for in-ear speech recognition [Kurcan, 2006] with a number of clusters K equal to 128 is used to obtain a 12 x 128 codebook. The feature vectors extracted during the sampling phase by the microprocessor (corresponding to the captured spoken word's samples) are used as input to the algorithm iterative calculations. A MATLAB implementation of the K-mean algorithm is used in this thesis, taken from [Kurcan, 2006]. The resultant codebook is converted into the 1.15 fractional format and hard-coded into the microprocessor's program memory.

The feature vector quantization procedure is executed in 256-sample frame base immediately after the feature extraction procedure. It is based on the calculation of the minimum Euclidean distance between a feature vector ($ft(i)$) and any of the 128 quantizer vectors ($cb(k,i)$), as given by Equation (7.1.1). The feature vector ft includes the second to the thirteen MFCC parameters previously extracted by the feature extraction procedure. Special microprocessor DSP commands are used to reduce the time required for quantization. All operations are performed in the 1.15 fractional format. The resultant code, representing a feature vector, is stored in a specific buffer. The algorithm is implemented in the "*_code_book_proc*" procedure.

$$code = index\ of\left[\min_{k=1:128}\left\{\sum_{i=1}^{12}\left(cb(k,i)-ft(i)\right)^2\right\}\right]. \tag{7.1.1}$$

Next an analytical discussion of the quantization procedure implemented in the real-time system is presented.

B. QUANTIZATION PROCEDURE

The quantization procedure (QP) is an implementation of Equation (6.1.1) into assembly language. The QP is characterized as a highly computational and demanding

routine in terms of floating point operations. It requires as many as 1536 subtraction, addition, and multiplication operations and 128 comparisons. To minimize the number of operations required for the QP the Euclidean distance between the vector components is calculated using the microprocessor's DSP error detection and correction (EDAC) core command. This command has the ability to execute a subtraction and a multiplication in one cycle and accumulate the result in the same cycle. Furthermore, the EDAC pre-fetches the values for the next operation. The QP algorithm was designed as follows:

1. Set i=1, *minimum* = $(7FFF)_{H,}$ and *minimum_position* = 0,
2. Take the *ft* vector and the *cb(i,:)* vector of the codebook,
3. a. Subtract, the corresponding components of the two vectors,
   b. Square and accumulate the result,
4. Compare the final accumulator context with the *minimum*,
5. Set the *minimum* equal to the final accumulator context if the latter is less than the former and set *minimum_position* equal to i,
6. Set $i = i + 1$,
7. Go to step two if $i < 128$,
8. Terminate the algorithm.

Steps 3.a and 3.b are executed in one cycle by using EDAC commands. The quantity *minimum* represents the minimum Euclidean distance while the quantity *minimum_position* corresponds to the index of the codebook code vector which had the minimum distance from the feature vector *ft*. The *minimum_position* is the discrete code that will represent the features vector *ft* in the recognition task.

The algorithm is implemented in assembly language as the "*_code_book_proc*" routine. The routine is activated by the feature extraction routine. The resultant code is an integer from one to 128 and is stored temporarily as a global variable, referred to as "*FrameFeatCode*," for use by the recognition routine. The algorithm takes a maximum of 4,000 pics to run.

Next, the real-time recognition procedure implementation is presented.

VIII. REAL-TIME ISOLATED WORD RECOGNITION

A. INTRODUCTION

As analyzed in Section II.C.4, a left-to-right, discrete-symbol, hidden Markov model (DHMM) with eight hidden states is the solution that best fits the real-time seven-word IWR system presented in this thesis. Figure 38 shows a typical eight-state, left to right, DHMM. Words are chosen as the speech utterances to be modeled. Thus, seven different DHMMs are implemented corresponding to the seven words of the system's vocabulary. The multiple observations scaling version of the *Baum-Welch* (B-W) training algorithm was applied to estimate the parameters $A^{(\lambda)}$, $B^{(\lambda)}$, $\pi^{(\lambda)}$ of each word model λ, as described in Section II.C.5.a. The parameters are converted into the 1.15 fractional format and are hard-coded into program memory in a specified order. Then, the recognition routine, with an input from a frame's feature vector code from the quantization step, calculates the log-likelihood for each model. The likelihood results are computed recursively from frame to frame by adding to the previous frame's likelihood estimates. Finally, the total log-likelihood of the observation (spoken word) occurrence for each model is calculated by the time the last frame is processed. The recognition result is the word which is represented by the model with the biggest log-likelihood. Figure 39 shows the block diagram of the seven-word IWR system.

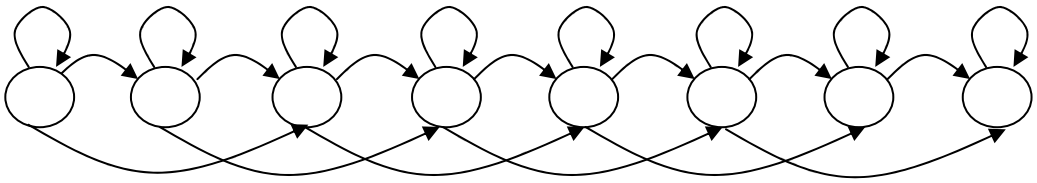

Figure 38. Left to Right 8-states Discrete Hidden Markov Model

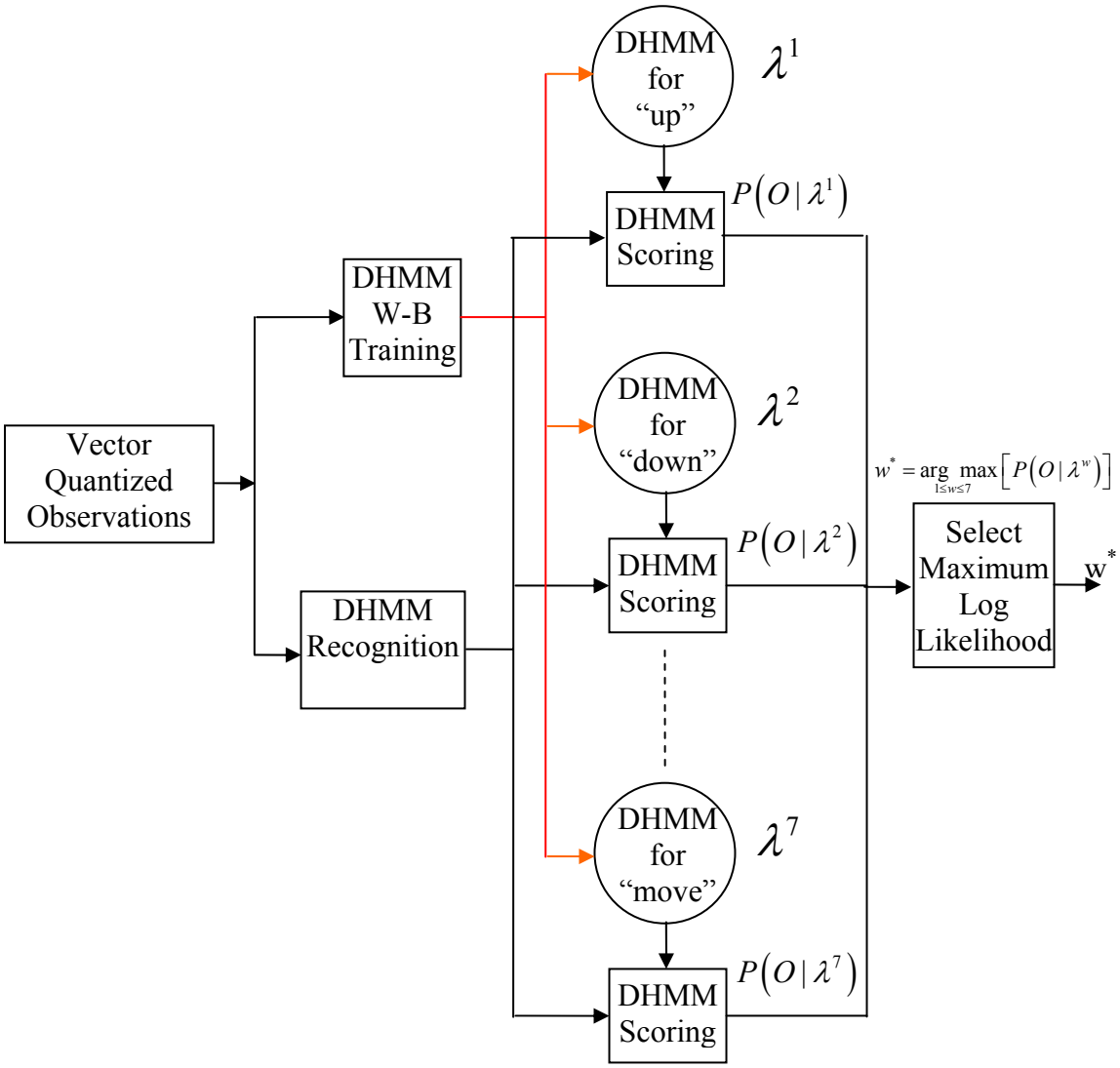

Figure 39. Block Diagram of the 7-word DHMM IWR Recognizer [After Kurcan, 2006].

Recall that the models' log-likelihood computations are executed on a frame by frame basis. As a result, the recognition decision is produced immediately after the word's end-point is detected and before the next 40-samples frame is captured. Consequently, the IWR system does not require any additional "silence" intervals between spoken words. The frame log-likelihood calculation (given by Equation 8.1.2) is based on the scale factor c_t^l given by Equation (8.1.1). It results from the Forward-Backward recursion algorithm, where t is a time index, here representing the current frame, and l is the model index, and $\alpha_t(i)$ is the forward variable defined as the

probability of occurrence a partial observation $\{o_1, o_2, ..., o_t\}$ given the state i in time t. Specific details regarding the algorithm may be found in [Kurcan, 2006].

$$c_t^l = \frac{1}{\sum_{j=1}^{S} \alpha_t^l(j)}, \quad \text{where} \quad \alpha_t(i) = P(o_1, o_2, ..., o_t \mid q_t = i, \lambda). \tag{8.1.1}$$

$$\log\left[P\left(O \mid \lambda^l\right)\right] = \sum_{t=1}^{T} \log c_t^l. \tag{8.1.2}$$

B. REAL-TIME IWR SYSTEM TRAINING

Training the real-time IWR system involves calculating the parameters $A^{(\lambda)}$, $B^{(\lambda)}$, $\pi^{(\lambda)}$ for each of the seven word's DHMM model $\lambda^{(l)}$. In this study, the B-W re-estimation algorithm for multiple observations is used, as discussed in Section II.C.4. A MATLAB implementation of this algorithm is used and is taken from [Kurcan, 2006]. Some changes in the program were required to adapt to the sampling procedure used in the real-time system. Specifically, the number of feature vectors (observations) per speech sample required to train the models was obtained by an off-line procedure in Kurcan's implementation, while in our study, the training algorithm is fed with feature vectors obtained by the capturing system and processed by the microprocessor's features extraction routines. The resultant parameters are of the following form:

- $A^{(\lambda)}$: seven arrays of size 8x8 of double precision numbers,
- $B^{(\lambda)}$: seven arrays of size 8x128 of double precision numbers,
- $\pi^{(\lambda)}$: seven arrays of size 1x8 of double precision numbers.

An executable program written in DELPHI V5 was developed to transform these parameters into 1.15 fractional binary numbers. The program output lists these binary numbers in hexadecimal form, a proper form for the microprocessor assembly compiler. Additionally, the arrays were reshaped into a one-dimensional matrix, as shown in Figure 40. This transformation was necessary, before the parameters are hard-coded in the microprocessor's program memory which requires one-dimensional structures. All algorithms were designed based on this parameter ordering scheme. Figure 40 illustrates an example of a parameter re-ordering, conversion, and storage into program memory. The operation instructions for the DELPHI program are given in Appendix C.

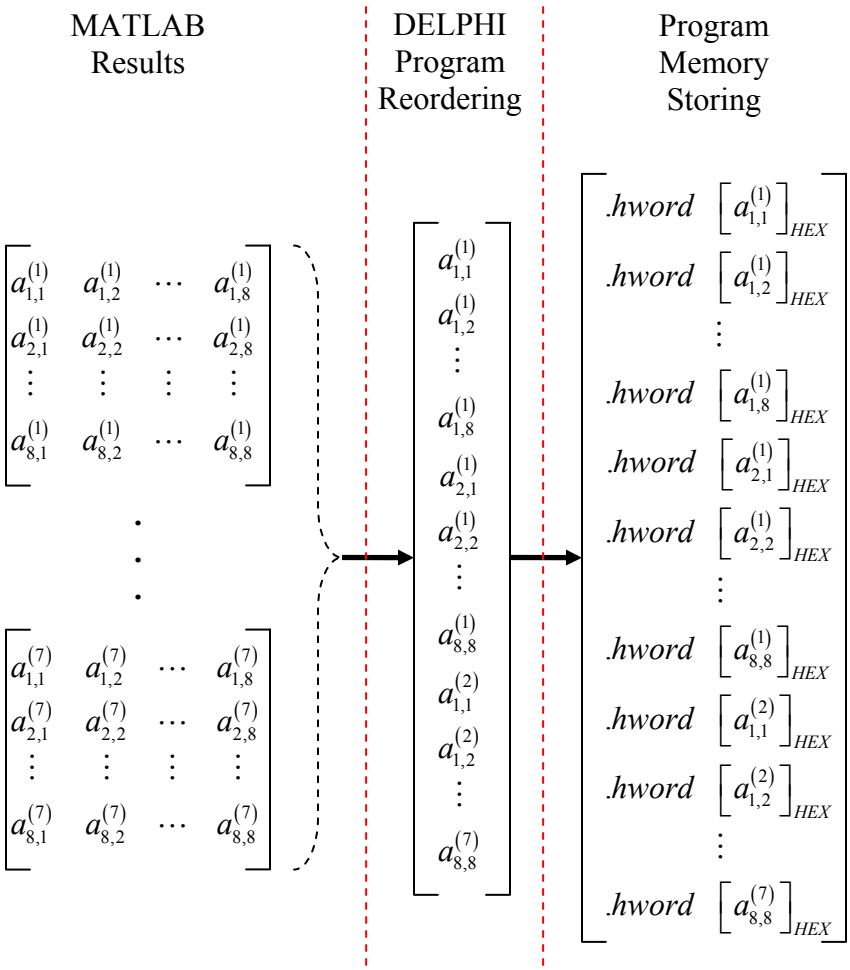

Figure 40. Example of A Parameter Re-ordering, Conversion, and Storage into Program Memory

C. REAL-TIME IWR SYSTEM SAMPLING

As described above, the model's training procedure requires multiple observations (samples). Ten persons of different ages, nine male and one female, contributed to the sampling procedure, speaking each word 50 times. A total of 500 samples for each word were used for the DHMM models training. The sampling procedure lasted 30 minutes for each person for each of the seven words.

The use of this study's capturing system and microprocessor for sampling was obviously a compulsory choice. Therefore, the parameters used for the IWR system must be obtained using the same methods and algorithms as those used during real-time recognition. Thus, a "*sampling*" option, which is activated by depressing button no. "2"

in the microprocessor board, was added to the program, as described in section IX.A. By choosing this option the system works identically as the basic program up to the features extraction step. Then, instead of directing the feature vectors to the quantization procedure (as described in Section VII.B) the program stores them in a buffer and continues capturing the next frame. In addition, the program stops to capture new samples when an end-point has been detected, and sends the stored features and the stored cropped spoken word's samples to a PC using the UART port (set as described in Section III.B). The features and the samples are captured by another application that was built in DELPHI V5. This application was responsible for receiving data from the UART, converting it from hexadecimal to double precision numbers and storing it in corresponding files named in accordance with a specific format. Three files are stored, one containing the features ("*.ftr*" extension), one containing the word's samples ("*.txt*" extension) and one containing the samples concatenated with the features in the original hexadecimal form that was received by the microprocessor ("*.hex*" extension). The name of each file is formatted taking four parameters into account:

- *Word's name*: Indicates the spoken word that corresponds to the sample.
- *Person Identity Number*: Indicates the person from whom the sample was taken and corresponds to thesis's human research database.
- *Sample's number*: Indicates the number of the sample.
- *Date*: Indicates the date the sample was taken.

Additionally, the DELPHI program is responsible for the creation of a directory tree into the "*C:/Thesis/*" directory. The tree consists of three directories "/samples", "/features" and "/Hex." In each of these directories, the program automatically creates directories for each person named by its identity number "/x". In these person's identity number directories, the program automatically creates new directories where the spoken samples are stored for each of the seven words ("/left"). The files were arranged in this fashion to achieve better control and execution speed during the MATLAB training procedure. An example of the sampling directory tree is shown in Figure 41.

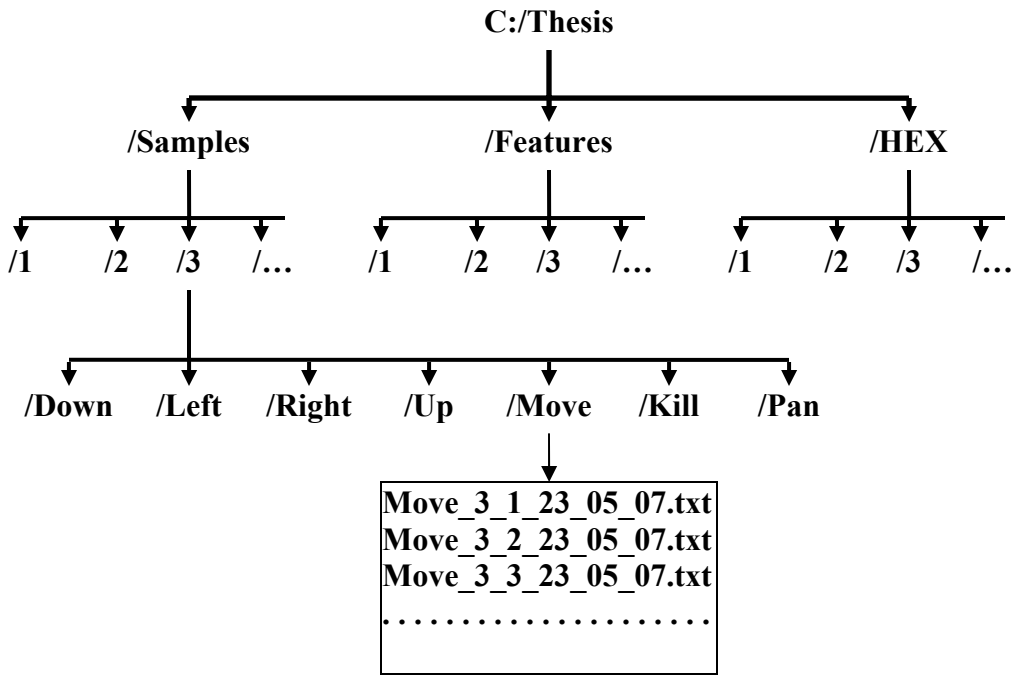

Figure 41. An Instance of the Sampling Directory Tree

Next we describe the recognition task as implemented in this study real-time IWR implementation.

D. REAL-TIME IWR IMPLEMENTATION

The IWR implementation is based on the Forward-Backward scoring algorithm for the calculation of the total probability P that an observation sequence O is produced by a DHMM $\lambda(P(O \mid \lambda))$. The algorithm is divided in two parts: the forward recursion and the backward recursion. The resultant P from the forward recursion is used for scoring and recognition, while the backward recursion is mainly useful for the training of the model [Kurcan, 2006]. To describe the three steps of the forward recursive algorithm, we introduce the forward variable $\alpha_t(i)$ as the joint probability of the partial observation sequence up to instant *t,* at the state *i* at the instant *t*, for a given model λ

$$\alpha_t(i) = P(o_1, o_2, ..., o_t \mid q_t = i, \lambda). \tag{8.4.1}$$

Next we obtain the scaled version $\bar{\alpha}_t(i)$ of the forward variable as:

$$\begin{aligned}\bar{\alpha}_t(i) &= c_t \alpha_t(i) \\ &= \frac{\alpha_t(i)}{\sum_{j=1}^{S} \alpha_t(j)},\end{aligned} \tag{8.4.2}$$

where $\alpha_t(i) = P(o_1, o_2, ..., o_t \mid q_t = i, \lambda)$, $c_t = \frac{1}{\sum_{j=1}^{S} \alpha_t(j)}$ is the scaling factor, and S represents the number of DHMM hidden states. The forward recursion is used to solve for $\alpha_t(i)$, given a DHMM λ with S number of hidden states, as follows [Kurcan, 2006]:

- Step 1: **Initialization:** for $1 \le i \le S$,

$$\alpha_1(i) = P(o_1, q_1 = 1 \mid \lambda) = \pi_i b(o_1 \mid q_1), \tag{8.4.3}$$

$$c_1' = \sum_{j=1}^{S} \alpha_1(j), \tag{8.4.4}$$

$$c_1 = \frac{1}{c_1'}, \tag{8.4.5}$$

$$\begin{aligned}\bar{\alpha}_1(i) &= c_1 \alpha_1(i) \\ &= \frac{\alpha_1(i)}{c_1'}.\end{aligned} \tag{8.4.6}$$

- Step 2: **Recursion:** for $t = 1, 2, ..., T-1$ and $1 \le j \le S$,

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{S} \bar{\alpha}_t(i) a(j \mid i) \right] b(o_{t+1} \mid j). \tag{8.4.7}$$

$$c_{t+1}' = \sum_{j=1}^{S} \alpha_{t+1}(j), \tag{8.4.8}$$

$$c_{t+1} = \frac{1}{c_{t+1}'}, \tag{8.4.9}$$

From Eqs (8.4.7) and (8.4.9)

$$\begin{aligned}\bar{\alpha}_{t+1}(j) &= c_{t+1}\alpha_{t+1}(j) \\ &= \frac{\alpha_{t+1}(j)}{c'_{t+1}}.\end{aligned} \tag{8.4.10}$$

- Step 3: **Termination:** for $1 \le i \le S$,

$$P(O \mid \lambda) = \sum_{i=1}^{S} \bar{\alpha}_{\mathrm{T}}(i) \tag{8.4.11}$$

From the definition of c_t (Eqs (8.4.4) and (8.4.5)) it follows that:

$$\begin{aligned}&\prod_{t=1}^{T} c_t P(O \mid \lambda) = 1, \text{ or} \\ &P(O \mid \lambda) = \frac{1}{\prod_{t=1}^{T} c_t}.\end{aligned} \tag{8.4.12}$$

Furthermore, taking the logarithm of the two sides yields the following log-likelihood expression [Deller, 2000; Kurcan, 2006]:

$$\log\left[P(O \mid \lambda)\right] = -\sum_{t=1}^{T} \log c_t. \tag{8.4.13}$$

Equation (8.4.13) gives the log-likelihood produced by the DHMM λ for a specific observation sequence O. This quantity is used to extract the final recognition result. The final result is obtained by comparing the log-likelihoods of all seven models for an observation, finding the maximum and declaring that the associated model corresponds to the spoken word.

The microprocessor implementation of the algorithm is the "*_log_likelihood_proc*" routine. The algorithm's three-step calculation is done on a frame by frame basis whenever a new quantized feature vector is available. Moreover, calculations are performed for all seven models. A new flag, "*IsFirstFrameFlg*", is introduced to distinguish the first frame (t=1) from the other frames (step one from step two). In each frame, the logarithm is calculated, in base two, of the resultant scale factor c'_t and it is added to the logarithm of the one produced by the previous frame. Therefore, we have log-likelihood estimates for each of the seven models immediately after the

calculations for the last frame. A final comparison determines the maximum among them and the model identity is forward to the robotic control part of the program. Simultaneously, the "*DCIDataAvailableFlag*" flag is set to declare that the recognition is finished.

The forward recursive algorithm equations above include a matrix with vector multiplication ([8x8]*[8x1]). Hence, we design a general matrix multiplication routine to ensure that all computations are performed in a 40-samples frame period. This routine is called "*_matrix_mult_8x8_8x1_proc.*" Furthermore, three new buffers are introduced, as follows:

- "*aaBuf*": Used for the storage of the forward variables $\bar{\alpha}_t^{(l)}(i)$, where *l* is the model index. Its length is equal to 56 words (7 models x 8 words/model = 56 words). The buffer is updated in a frame base.
- "*ATempBuf*": Used to store a copy of the $A^{(\lambda)}$ parameters array of the current model from the program memory to data memory. Its length is 64 words (8x8 words). It is updated based on the current model.
- "*PmTempBuf*": Used to store a copy of the $\pi^{(\lambda)}$ parameters array of the current model from the program memory to data memory. Its length is eight words (1x8 words). It is updated based on the current model.

The recognition routine runs up to 8,400 pics per frame. The total processing time needed for each 256-sample overlap frame, including end-point detection, features extraction, and recognition routines, is approximately **34,850** pics.

Next, we present the end-to-end system operation and evaluation of the human-machine robotic control interface based on the ear-microphone IWR.

THIS PAGE INTENTIONALLY LEFT BLANK

IX. REAL-TIME IWR SYSTEM OPERATION - EVALUATION

This chapter presents the real-time IWR system operation from the user's perspective. Furthermore, we describe the sampling and evaluation methods that are used to determine system performances, and present performance results.

A. REAL-TIME IWR SYSTEM OPERATION

Two different operation modes are available in a real-time IWR system. The first is the "*main*" operation mode in which the system is setup and operates in a real-time IWR mode in human-machine interface for robotic control or other. The second is the "sampling" operation mode that is used to obtain the spoken word samples necessary for the training of the DHMM word models used in this study. These two different operation modes are described thoroughly in the following sections.

1. Real-time IWR System "*Main*" Operation Mode

The "*main*" operation mode refers to the use of the system as a real-time IWR system, using an ear-microphone as a human-machine interface for robotic control. The implementation of this mode, in this study, requires that a set of actions be performed by the user (speaker) to operate the real-time IWR system. These steps can be divided into two phases; the initialization phase and the normal operation phase. The system initialization phase includes the following actions by the user:

1. Placement of the ear-microphone on the speaker,
2. Connection of the microphone connector into the audio codec device,
3. Connection of the audio codec device with the microprocessor board,
4. Connection of the microprocessor control output ports with the machine controller,
5. Setting of the microprocessor power supply to the 'ON' position.

Two LEDs (as depicted in Figure 42) are used to indicate that initialization has been successfully completed: the LED no. "1" located on the microprocessor board and the audio codec LED. The former must be 'ON' while the later illuminates periodically. Any other configuration of LED illumination indicates a malfunction of the system. In this case the user must reset the system by turning the power supply 'OFF' and 'ON.'

After initialization, the system is ready to operate as a real-time IWR system. The user depresses the button no. “1” on the microprocessor board (Figure 42). The LED “1” on the microprocessor board goes ‘OFF’ once and remains ‘ON’ indicating that the threshold calculations have been completed. Then, the user can speak commands. After a command is spoken, the system immediately captures the speech signal via the ear-microphone and produces the recognition results to the user. LEDs “2,” “3,” “4,” and “5” (Figure 42) are used to interpret the binary code for each of the recognized command described in Table 3.1. The LEDs indicate the recognized spoken word and change only when a new command is recognized (the LEDs transition to ‘OFF’ just before the indication of a new recognized word).

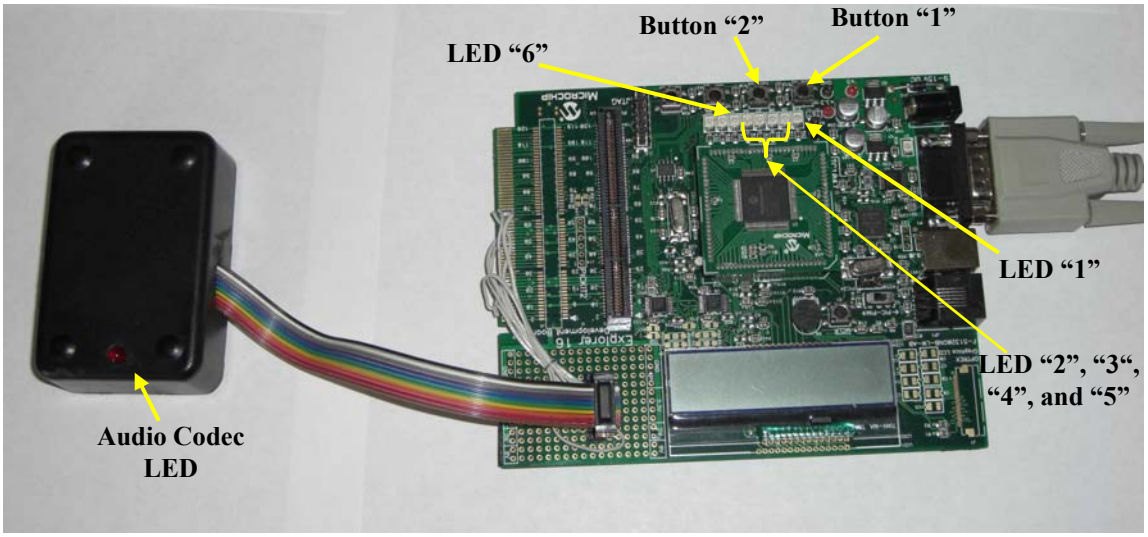

Figure 42. Real-time IWR System Operation Buttons and LED

At this time, the real-time IWR system does not distinguish between regular speech and an actual robot command (e.g. a user speaking one of the 7 words in conversation as opposed to direct a robot). Thus, a push-to-talk (PTT) button was added to the microphone. Using the PTT button, the user is able to control when his spoken words are intended as commands for the IWR system.

In addition to the normal recognition event, two more events can occur during the operation of the real-time IWR system; a no-recognition event and a general error event.

The first event occurs when the system is unable to recognize the spoken word. This event is indicated to the user by LED code "1001", and does not affect system operation. In this case, the system continues to capture speech and the user must repeat the command. The second event occurs when an address error, an arithmetic error, an oscillator error, or a stack error is encountered by the microprocessor. This type of event is fatal for the system. It is indicated by LED code "1111". In this case, the user must reset the system by powering it 'OFF.' It should be noted that this event rarely occurs. Multiple instances of this event indicate a serious problem with the microprocessor, requiring the user to reprogram the microprocessor or replace it.

2. Real-time IWR System "Sampling" Operation

The "sampling" operation mode refers to the use of the system to obtain the spoken word samples necessary to train the DHMM word models used in this study. The samples are obtained using the real-time IWR system, running on the microprocessor, in conjunction with another application, running on a computer, as described in Section VIII.C. Thus, a set of actions in both applications (IWR in microprocessor and DELPHI application in the computer) must be performed by the user (speaker) to obtain these samples. These steps may be divided in two phases; the initialization phase and the normal operation phase. The system initialization phase includes the following actions by the user:

1. Placement of the ear-microphone on the speaker,
2. Connection of the microphone connector into the audio codec device,
3. Connection of the audio codec device with the microprocessor board,
4. Connection of the microprocessor's serial port with the computer serial port,
5. Setting of the microprocessor power supply to the 'ON' position,
6. Execution of the "*IWR_Sampling_Appl.exe*" on the computer,
7. Selection of the word to be sampled, and identification of the number of the speaker using the dialog box of Figure 43.

After this last step, the computer application is ready to receive samples as depicted in Figure 44. Simultaneously, two LEDs are used to indicate that the microprocessor's initialization has been successfully completed: the LED "1" on the microprocessor board and the audio codec LED, as previously depicted in Figure 42. The former must be 'ON' while the later illuminates periodically. Any other configuration of

LED illumination indicates a malfunction of the system. In this case the user must reset the system by turning the power supply 'OFF' and 'ON'.

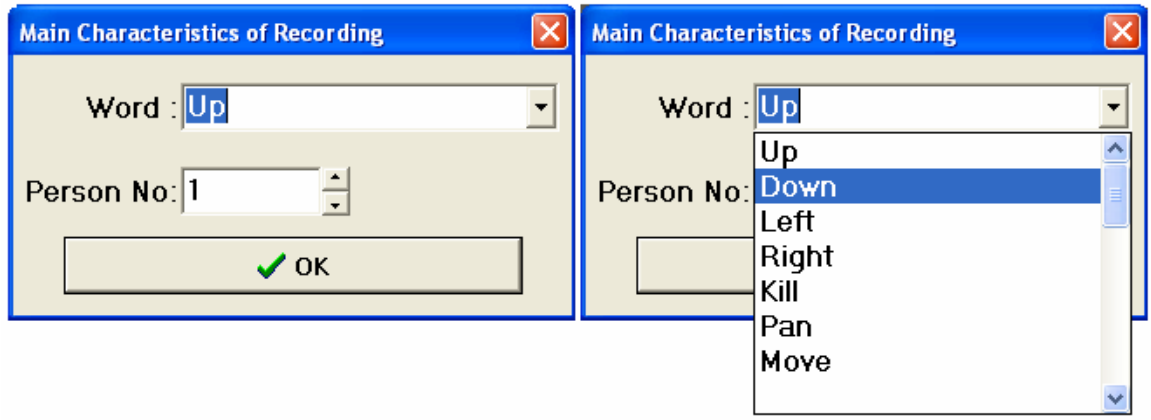

Figure 43. Dialog Box for Choosing the Word and the Speaker Identity Number of the current Spoken Word's Sample

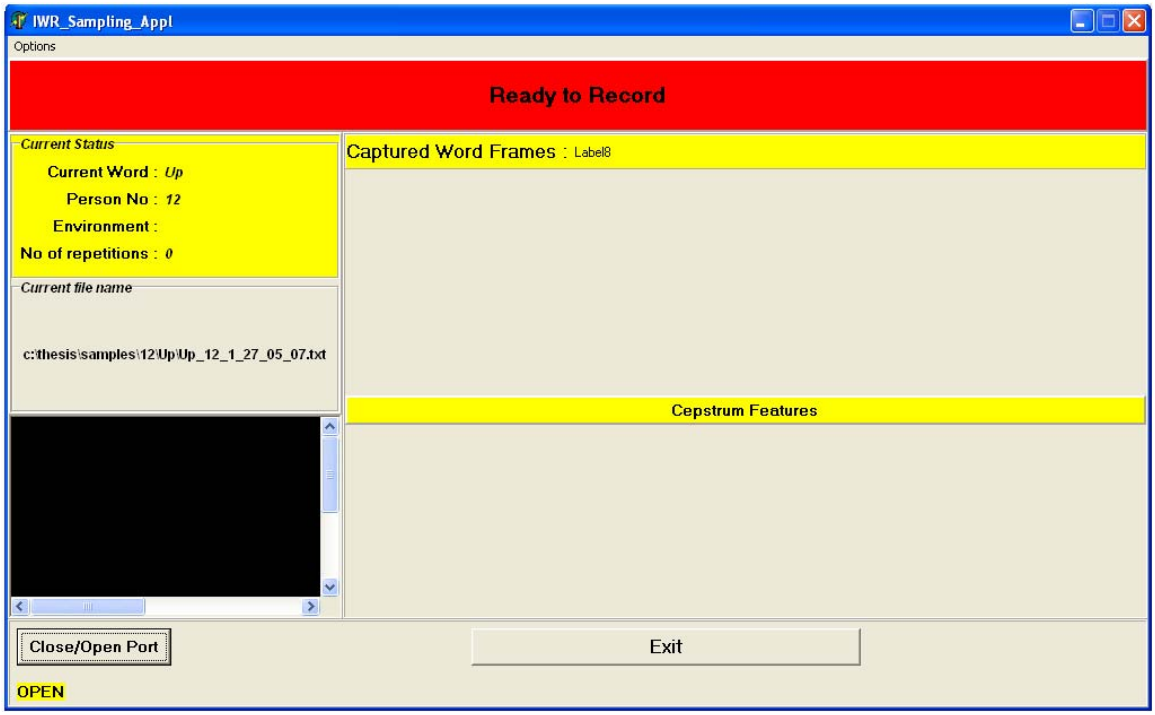

Figure 44. *"IWR_Sampling_Appl.exe"* Application's Main Window. The application has been initialized and is ready to record the data received from the microprocessor

After initialization, the system is ready to capture speech samples or to operate as a real-time IWR system. To activate the "sampling" mode of the system, the user depresses button no. "2" on the microprocessor board (Figure 42). LED "1" on the microprocessor board goes 'OFF' once and remains 'ON', indicating that the threshold calculations have been completed and the user can begin sampling. Every time a spoken word is captured the computer application acknowledges the data reception to the microprocessor. In addition, the application shows in two different plots the captured spoken word sample and its features as obtained by the microprocessor (Figure 45), and activates a dialog box (Figure 45) requesting the user to choose among the following options:

1. "*Next Repetition;*" Save the captured sample in the proper directory and file format as described in Section VIII.C, update the sample's trial number and wait for the next sample,
2. "*Cancel Previous Repetition;*" Discard the received data and wait for a new trial of the same sample to be captured,
3. "*Change Word or Person;*" Save the captured sample in the proper directory and file format as described in Section VIII.C, show the dialog box for choosing the sampling word and speaker's identity number, as shown n Figure 43. Set the trial number to one and keep the new chosen word and speaker identity values when the user finally confirms a choice,
4. "*Stop;*" Save the last captured sample in the proper directory and file format as described in Section VIII.C, and terminate the application.

Furthermore, labels are included in the computer application to show the current status of the sampling procedure such as the current sampled word and speaker's identification number, the full path of the file name that the sample word is stored, the current spoken word's 256-sample overlap frames, and the number of repetitions up to this point, as depicted in Figure 45. The time required for sampling 50 samples of a spoken word is less than five minutes.

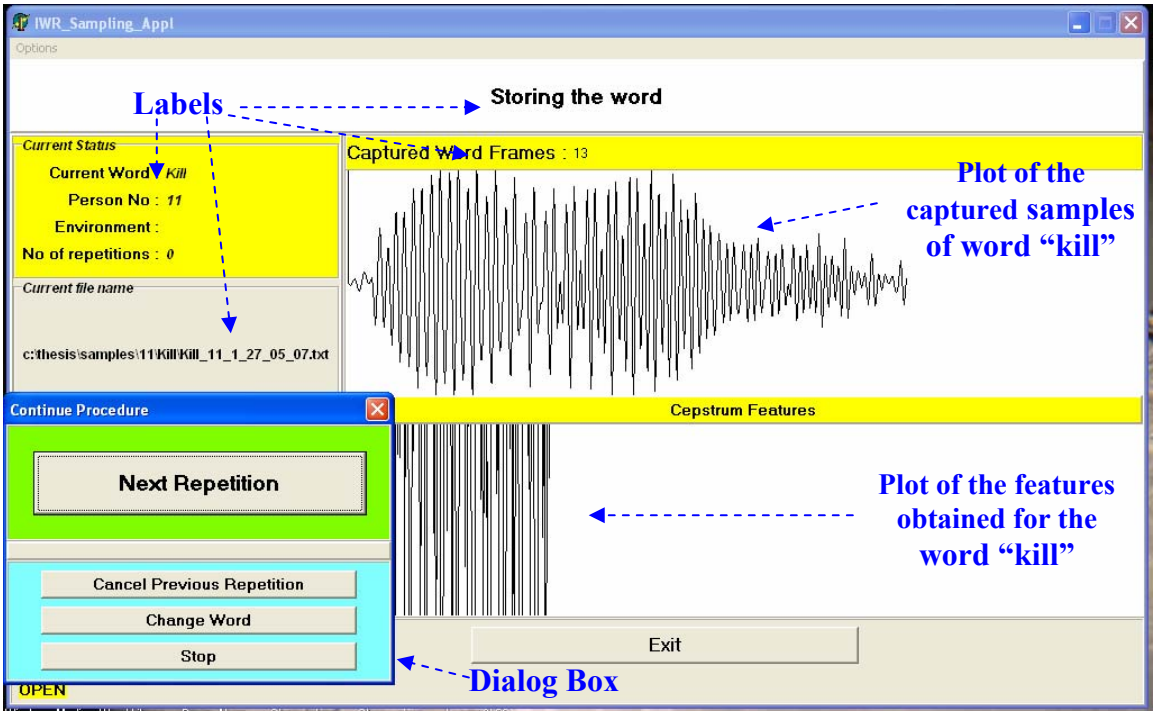

Figure 45. *"IWR_Sampling_Appl.exe"* Application's window after receiving the spoken word's samples from the microprocessor. A dialog box is open waiting for user Choice. Previously captured samples and features have been plotted. The labels have been updated indicating the current situation of the application.

Next we discuss the evaluation methods used in this study to obtain the real-time IWR system performance results.

B. REAL-TIME IWR SYSTEM EVALUATION

Three different approaches were used to evaluate the performance of the real-time IWR system; the off-line method, the real-time speaker-dependent method, and the real-time speaker-independent method. The first method is performed by taking arbitrary training samples, feeding them into the real-time IWR system via the UART port, and computing the recognition results of the system. The second is performed in real-time by using the speakers employed in the training phase, to speak each of the seven commands and then computing the system recognition results. The third method is similar to the second one but with speakers that did not participate in the training phase used in the testing phase.

1. Off-line IWR System Evaluation

Five different experiments were performed using this method. The overall classification rate is **75.28%.** Table 8.1 depicts the confusion matrix of the average recognition results, while Figure 46 shows the average recognition results per word. Each experiment was set up by choosing arbitrary half of the available samples to train the DHMM recognizer and the rest to evaluate the real-time IWR system performance.

	AVERAGE RECOGNITION RATES FOR THE OFF-LINE IWR EVALUATION METHOD (%)						
WORD	Up	Down	Left	Right	Kill	Pan	Move
Up	**80**	1	0	0.5	0	0	0
Down	10	**74**	3	7	1	5	6
Left	5.5	2.5	**71.5**	2.5	7.5	1	3.5
Right	3.5	7	7	**66**	5	3	10.5
Kill	0	0.5	2.5	5.5	**70.5**	1.5	1.5
Pan	0	13.5	6	1.5	3	**87**	0.5
Move	1	1.5	10	17	13	2.5	**78**
Average Recognition Rate: **75.28 %**							

Table 9.1 Average Recognition Rate Confusion Matrix; Off-line Evaluation Method

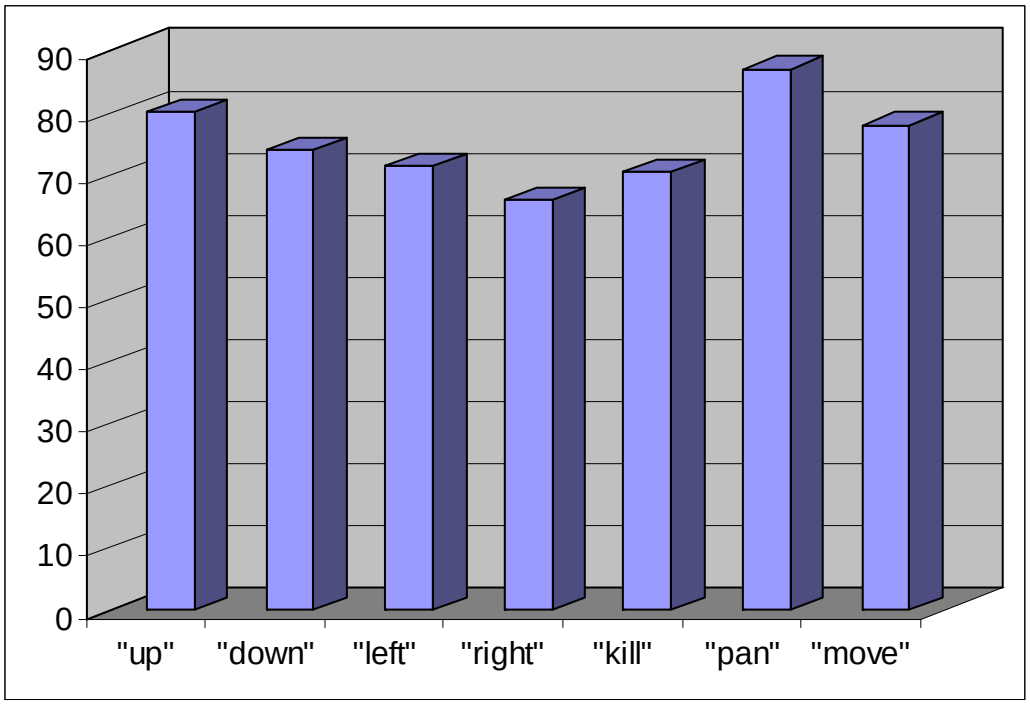

Figure 46. Average Recognition Results per Word; five Experiments Using the Off-line Evaluation Method

2. Speaker-dependent Real-time Evaluation Method

Five different experiments were also conducted using this method, with two different speakers. The resulting overall classification rate is **75.08%**. Table 8.2 depicts the confusion matrix obtained for average recognition results, while Figure 47 shows the average recognition results per word. Each experiment was set up by choosing 25 arbitrary samples out of 50 per word and per speaker to train the DHMM recognizer. Then, two different speakers spoke the seven-word commands for 20 repetitions each. The speakers used in these experiments participated in the training phase. The real-time IWR system LED indicators were used to record the recognition results.

	AVERAGE RECOGNITION RATES FOR THE REAL-TIME SPEAKER-DEPENDENT EVALUATION METHOD (%)						
WORD	Up	Down	Left	Right	Kill	Pan	Move
Up	**82.8**	1.2	0	0.4	0	0	0
Down	8.4	**75.2**	2.8	9.6	1.2	6	7.2
Left	4.4	2.8	**70**	2.8	6.4	2.8	4
Right	2.8	6	7.6	**64**	4.4	2.4	9.6
Kill	0	1.2	4.8	6.8	**72.8**	2	1.2
Pan	0	12	5.6	1.2	3.2	**84**	1.2
Move	1.6	1.6	9.2	15.2	12	2.8	**76.8**
Average Recognition Rate: **75.08 %**							

Table 9.2 Average Recognition Results Confusion Matrix; Real-time Speaker-dependent Evaluation Method

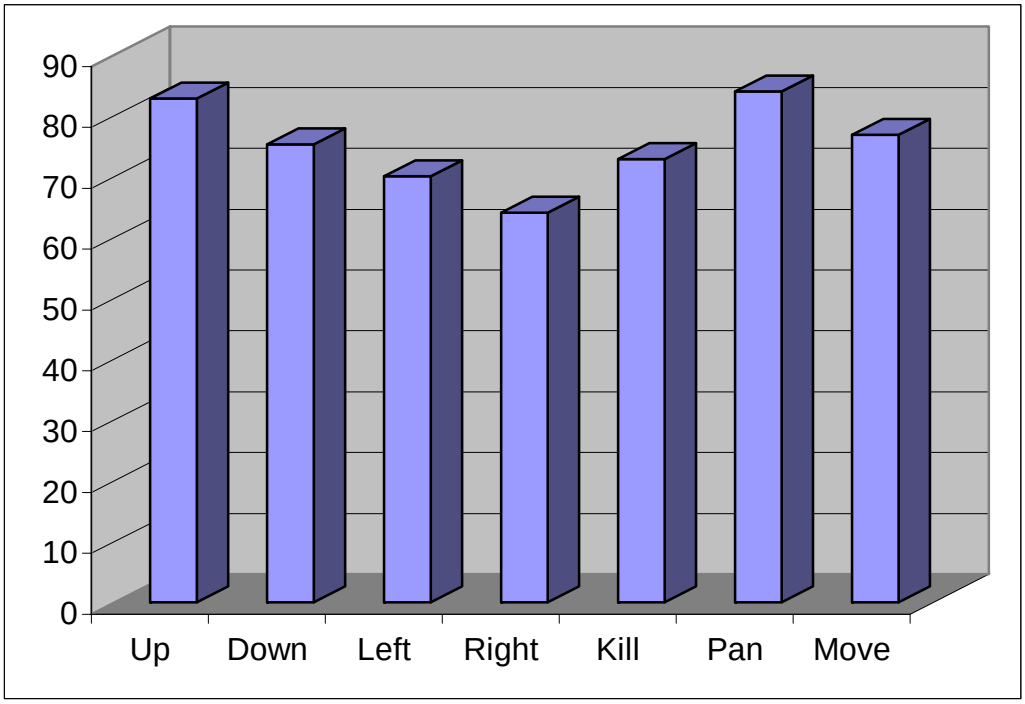

Figure 47. Average Recognition Results per Word; five Experiments Using the Real-time Speaker-dependent Evaluation Method

3. Speaker-independent Real-time Evaluation Method

Five different experiments using this method were also conducted, using two different speakers. The resulting overall classification rate is **70.82%**. Table 8.3 depicts the confusion matrix of the average recognition results, while Figure 48 shows the average recognition results per word. Each experiment was setup by choosing 25 arbitrary samples out of 50 per word and per speaker to train the system's DHMM. Then two new speakers not previously used in the training phase spoke the seven-word commands for 20 repetitions each. The real-time IWR system LED indicators were used to compute the recognition results.

	AVERAGE RECOGNITION RATES FOR THE REAL-TIME SPEAKER-DEPENDENT EVALUATION METHOD (%)						
WORD	Up	Down	Left	Right	Kill	Pan	Move
Up	**78.4**	2.2	1.2	0.4	0.2	1	0
Down	9.2	**70.6**	3.4	12.2	1.8	5.4	5.2
Left	6	2.4	**64**	3	8.4	3.6	7
Right	2.4	4.8	8.4	**62.4**	4	2	12.4
Kill	1	2.4	4.2	4.2	**69.2**	3.2	2.6
Pan	1.2	15	4.6	1.6	3.8	**80**	1.6
Move	1.8	2.6	14.2	16.2	12.6	4.8	**71.2**
Average Recognition Rate: **70.82 %**							

Table 9.3 Average Recognition Results Confusion Matrix; Real-time Speaker-independent Evaluation Method

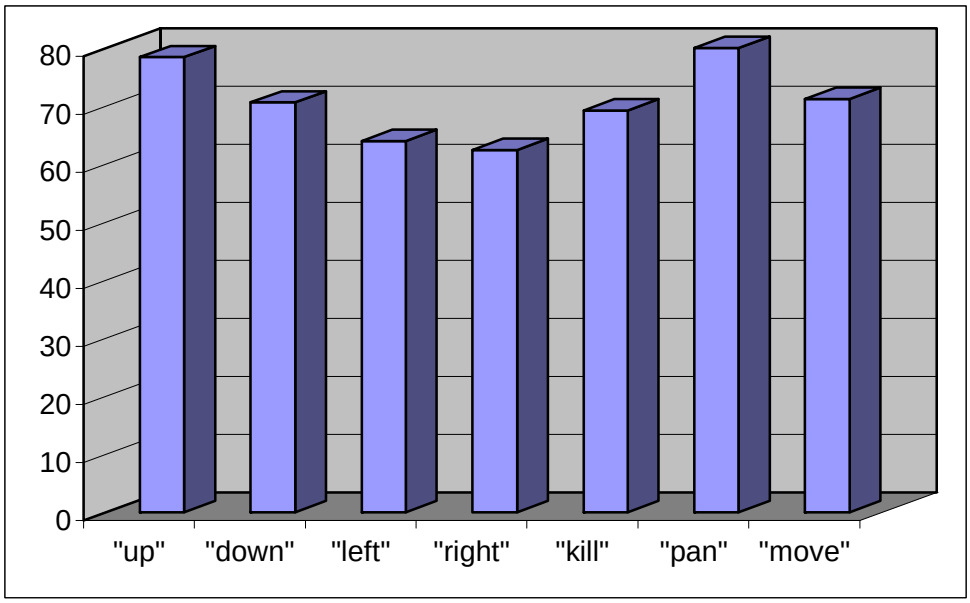

Figure 48. Average Recognition Results per Word; Five Experiments Using the Real-time Speaker-independent Evaluation Method

The resulting overall average recognition rate for the system is **73.72%**. In the next chapter, the results obtained from the evaluation experiments are used to evaluate the total performance of this study's real-time IWR system implementation. Furthermore, we discuss the challenges that were overcome during this research. Finally, we present recommendations for improvements and extensions to this research.

X. CONCLUSIONS

This study implemented a portable COST low cost real-time isolated word recognition system (IWR) based on an ear-microphone as a human-machine interface for robotic control applications. The main advantage of this system is that it enables hands-free control increasing a user's operational capabilities, while the ear microphone is well suited to noisy environments as the ear canal shields the collected speech from environmental noises.

A. SIGNIFICANT RESULTS AND CONCLUSIONS

This study showed that a real-time implementation of an IWR system based on an ear-microphone is feasible. The following comments can be made regarding the hardware and software implementations:

- **Hardware**
  - The speech capturing system, as implemented using the ear-microphone XEM98D and the Si3000 Audio Codec, captures, amplifies, filters, and samples speech clearly, without adding any significant amount of processing noise. In addition, the ear-microphone can shield the speaker voice from environmental noise.
  - The Microchip dsPIC33FJ256GP710 microprocessor used for implementation of the real-time IWR executes the processing routines in real-time without stalls or errors. The microprocessor's I/O devices (DCI, UART) work effectively with the audio codec and computer used for sampling and training, without communication problems or data loss. Its data and program memory capacity is more than adequate for this study's implementation. Finally, the microprocessor's DSP core's operation and available commands make the execution of complex routines faster and more accurate achieving good recognition performance and leaving available time for command and control tasks.
  - The Microchip Explorer 16 demo-board offered a platform that is adequate for the programming and evaluation of the IWR system. Furthermore, it offers the necessary hardware for implementing a command and control interface.
- **Software**
  - The whole program executes in real-time as was designed without errors. Specifically, the program never crashed, even after many hours of operation.

- The end-point detection routines, as implemented in this study cropp the spoken words accurately and in real-time.
- The feature extraction routines, as implemented in this study, process captured spoken word samples in real-time without loosing any significant amount of information.
- The recognition routines, as implemented in this study, recognize spoken word in real-time with sufficient performance. This feature is particularly notable considering that this implementation is the first step in transforming an "off-line" theoretical model into an actual real-time model.
- The robotic control routine, as implemented in this study, is capable of controlling a machine in real-time.
- The current amount time allocated for each frame capture is equal to 147,500 pics resulting from the end-point detection procedure, as described in Chapter IV. The total processing time for the seven-word vocabulary considered in this study (including end-point detection, feature extraction, and recognition routines) is approximately equal to 34,850 pics, leaving an additional 112,650 pics for control and command tasks (147,500 – 34,850 = 112,650).

B. RECOMMENDATIONS FOR FUTURE WORK

The real-time IWR system based on an ear-microphone implemented in this study is the first step towards transforming a theoretical model into a real-world device. However, it was not possible to examine all issues that may affect performances due to the multiples aspects covered in the study (hardware design and construction, software analysis, design and implementation, sampling, training, evaluation, thesis writing) in the allotted amount of time for completion Thus, the following issues remain open for consideration:

- **Hardware**
  - The microprocessor used in our study has 16-bit arithmetic. The use of a 32-bit arithmetic logic microprocessor would increase the recognition results as it would give more accurate arithmetic results during feature extraction and recognition routines.
- **Software**
  - The routine used for calculation of the logarithm of an integer number can be improved to give more accurate results. The logarithm procedure is critical in the accuracy of the feature extraction and likelihood calculations affecting the system recognition performance. Simulations showed the recognition rate obtained using features derived from the same sampled

data using floating-point precision to be around 97% as compared to 75.28% for the off-line implementation reported in Section IX.C.1 (Table 9.1), which indicates the negative impact the arithmetic precision had on the system performances.

- The current implementation does not make use of the entire time available for command-and-control tasks, as mentioned in Section III.B.4. Thus the recognition scheme could be further enhanced to improve resulting recognition rates. A more complex routine could be developed for specific robotic applications, such as the control of Unmanned Air Vehicles (UAVs) for example.

- **Theoretical Model**
  - Different aspects of the theoretical model can be evaluated using this study's real-time IWR software and hardware platform. First, investigating how the recognition performances are affected by changing the number of hidden states used in the DHMM could be conducted. Second, the dynamic Delta-MFCC parameters could be added to introduce time-varying information in the word models [Gold, 2000; Kurcan, 2006].

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. MICROCHIP DSPIC33FJ256GP710 MICROPROCESSOR GENERAL SPECIFICATIONS [MICROCHIP, 2006]

1. SPECIFICATION

- **Operating Range:**
  - DC – 40 MIPS (40 MIPS @ 3.0-3.6V, -40°C to +85°C)
  - Industrial temperature range (-40°C to +85°C)
- **High-Performance DSC CPU:**
  - Modified Harvard architecture
  - C compiler optimized instruction set
  - 16-bit wide data path
  - 24-bit wide instructions
  - Linear program memory addressing up to 4M instruction words
  - Linear data memory addressing up to 64 Kbytes
  - 83 base instructions: mostly 1 word/1 cycle
  - Sixteen 16-bit General Purpose Registers
  - Two 40-bit accumulators:
    - With rounding and saturation options
  - Flexible and powerful addressing modes:
    - Indirect, Modulo and Bit-Reversed
  - Software stack
  - 16 x 16 fractional/integer multiply operations
  - 32/16 and 16/16 divide operations
  - Single-cycle multiply and accumulate:
    - Accumulator write back for DSP operations
    - Dual data fetch
  - Up to ±16-bit shifts for up to 40-bit data
- **Direct Memory Access (DMA):**
  - 8-channel hardware DMA:
  - 2 Kbytes dual ported DMA buffer area (DMA RAM) to store data transferred via DMA:

- Allows data transfer between RAM and a peripheral while CPU is executing code (no cycle stealing)
- Most peripherals support DMA

- **Interrupt Controller:**
  - 5-cycle latency
  - 118 interrupt vectors
  - Up to 67 available interrupt sources
  - Up to 5 external interrupts
  - 7 programmable priority levels
  - 5 processor exceptions
- **Digital I/O:**
  - Up to 85 programmable digital I/O pins
  - Wake-up/Interrupt-on-Change on up to 24 pins
  - Output pins can drive from 3.0V to 3.6V
  - All digital input pins are 5V tolerant
  - 4 mA sink on all I/O pins
- **On-Chip Flash and SRAM:**
  - Flash program memory 256 Kbytes
  - Data SRAM, up to 30 Kbytes (includes 2 Kbytes of DMA RAM)
- **System Management:**
  - Flexible clock options:
    - External, crystal, resonator, internal RC
    - Fully integrated PLL
    - Extremely low jitter PLL
  - Power-up Timer
  - Oscillator Start-up Timer/Stabilizer
  - Watchdog Timer with its own RC oscillator
  - Fail-Safe Clock Monitor
  - Reset by multiple sources
- **Power Management:**
  - On-chip 2.5V voltage regulator
  - Switch between clock sources in real time

- Idle, Sleep and Doze modes with fast wake-up

- **Timers/Capture/Compare/PWM:**
  - Timer/Counters, up to nine 16-bit timers:
    - Can pair up to make four 32-bit timers
    - 1 timer runs as Real-Time Clock with external 32.768 kHz oscillator
    - Programmable prescaler
  - Input Capture (up to 8 channels):
    - Capture on up, down or both edges
    - 16-bit capture input functions
    - 4-deep FIFO on each capture
  - Output Compare (up to 8 channels):
    - Single or Dual 16-Bit Compare mode
    - 16-bit Glitchless PWM mode
- **Communication Modules:**
  - 3-wire SPI (up to 2 modules):
    - Framing supports I/O interface to simple codecs
    - Supports 8-bit and 16-bit data
    - Supports all serial clock formats and sampling modes
  - I2C™ (up to 2 modules):
    - Full Multi-Master Slave mode support
    - 7-bit and 10-bit addressing
    - Bus collision detection and arbitration
    - Integrated signal conditioning
    - Slave address masking
  - UART (up to 2 modules):
    - Interrupt on address bit detect
    - Interrupt on UART error
    - Wake-up on Start bit from Sleep mode
    - 4-character TX and RX FIFO buffers
    - LIN bus support
    - IrDA® encoding and decoding in hardware
    - High-Speed Baud mode
    - Hardware Flow Control with CTS and RTS
  - Data Converter Interface (DCI) module:
    - Codec interface
    - Supports I2S and AC'97 protocols
    - Up to 16-bit data words, up to 16 words per frame
    - 4-word deep TX and RX buffers
  - Enhanced CAN (ECAN™ module) 2.0B active (up to 2 modules):

    - Up to 8 transmit and up to 32 receive buffers
    - 16 receive filters and 3 masks
    - Loopback, Listen Only and Listen All Messages modes for diagnostics and bus monitoring
    - Wake-up on CAN message
    - Automatic processing of Remote Transmission Requests
    - FIFO mode using DMA
    - DeviceNet™ addressing support
- **Motor Control Peripherals:**
    - Motor Control PWM (up to 8 channels):
        - 4 duty cycle generators
        - Independent or Complementary mode
        - Programmable dead time and output polarity
        - Edge or center-aligned
        - Manual output override control
        - Up to 2 Fault inputs
        - Trigger for ADC conversions
        - PWM frequency for 16-bit resolution (@ 40 MIPS) = 1220 Hz for Edge-Aligned mode, 610 Hz for Center-Aligned mode
        - PWM frequency for 11-bit resolution (@ 40 MIPS) = 39.1 kHz for Edge-Aligned mode, 19.55 kHz for Center-Aligned mode
    - Quadrature Encoder Interface module:
        - Phase A, Phase B and index pulse input
        - 16-bit up/down position counter
        - Count direction status
        - Position Measurement (x2 and x4) mode
        - Programmable digital noise filters on inputs
        - Alternate 16-bit Timer/Counter mode
        - Interrupt on position counter rollover/underflow
- **Analog-to-Digital Converters (ADCs):**
    - Up to two ADC modules in a device
    - 10-bit, 1.1 Msps or 12-bit, 500 Ksps conversion:
        - 2, 4 or 8 simultaneous samples
        - Up to 32 input channels with auto-scanning
        - Conversion start can be manual or synchronized with 1 of 4 trigger sources
        - Conversion possible in Sleep mode
        - ±2 LSb max integral nonlinearity
        - ±1 LSb max differential nonlinearity
- **CMOS Flash Technology:**
    - Low-power, high-speed Flash technology

- Fully static design
- 3.3V (±10%) operating voltage
- Industrial temperature
- Low-power consumption

- **Packaging:**
  - 100-pin TQFP (14x14x1 mm and 12x12x1 mm)
  - 80-pin TQFP (12x12x1 mm)
  - 64-pin TQFP (10x10x1 mm)

2. GENERAL BLOCK DIAGRAM [MICROCHIP, 2006]

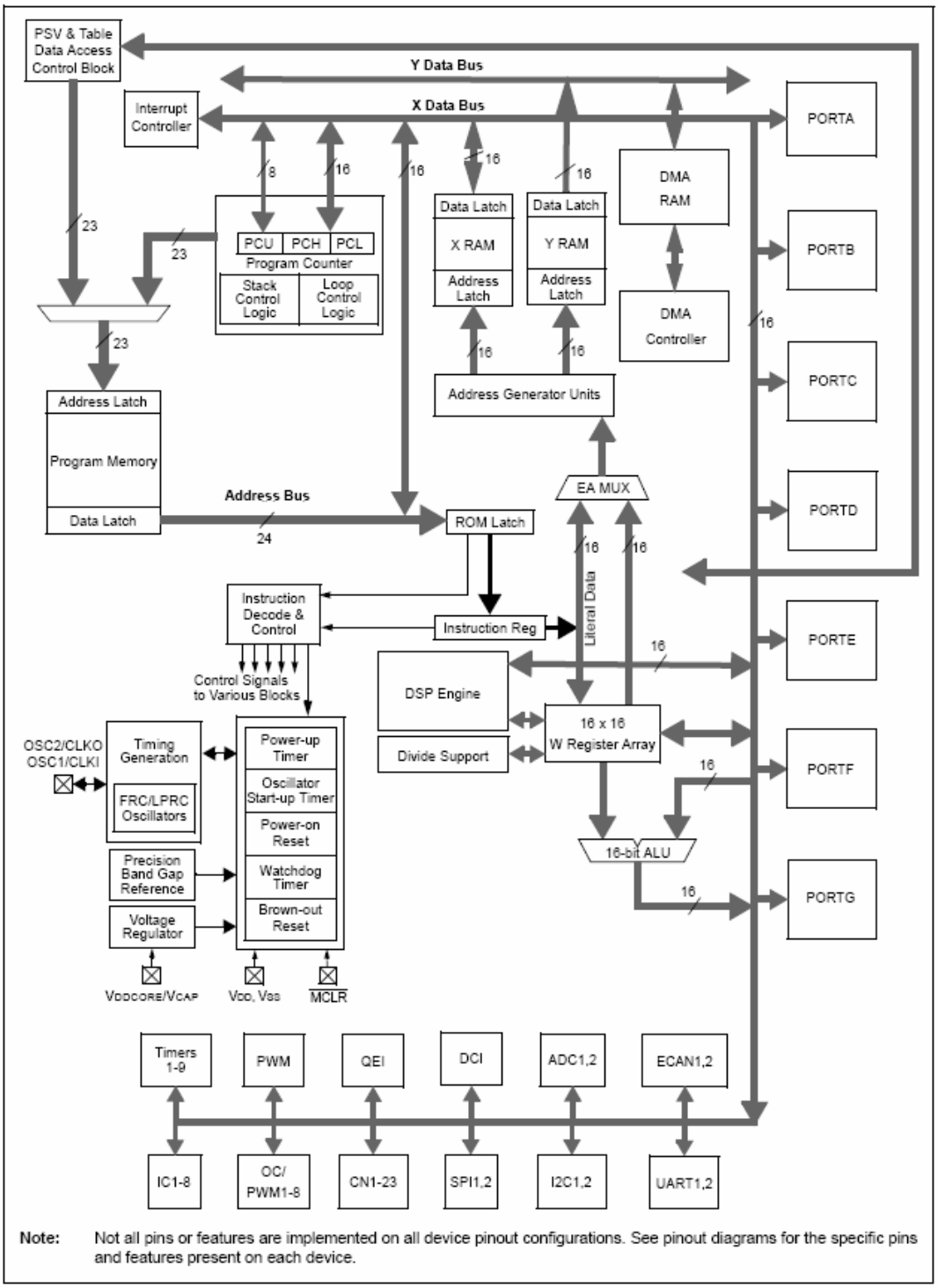

Figure 49. Microchip dsPIC33FJ256GP710 General Block Diagram [Microchip, 2006]

APPENDIX B. AUDIO CODEC SI3000 REGISTER SETUP

The Si3000 Audio Codec by Silicon Laboratories uses nine 8-bit registers to control its operation. The setup of these registers used in this study follows:

Register	Value	Si3000 Operation
Control Register 1	[0001 0000]	Normal operation using the microphone input and the speakers output.
Control Register 2	[0000 0000]	The digital filtering is enabled and the PLL divider is set to 5.
PLL 1 Divide N1 Register	[0000 0000]	N1 Phase locked loop (PLL) divider is set to 1 (0+1).
PLL 1 Multiply M1 Register	[0001 0011]	M1 Phase locked loop (PLL) multiplier is set to 20 (19 +1).
Rx Gain Control Register	[0111 0010]	Line in and Handset inputs are muted, the Microphone pre-amplifier Gain is 20 dB, and an FIR digital filter is active.
ADC Volume Control Register	[0101 0000]	A/D converter volume is -4.5 dB. Line Out and Headset outputs are muted.
DAC Volume Control Register	[0111 1111]	12 dB gain for the speaker output. Left and right speaker outputs are active.
Analog Attenuation Register	[0101 1100]	0 dB analog attenuation in speakers output
Status Report Register		Is Read-only register.

Table B.1 Setup of Si3000 Audio Codec Registers

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C. PARAMETERS NUMERICAL CONVERSION APPLICATION

The "*Param_Conv_1_15_Fract_App.exe*" application converts the parameters used in this study to the 1.15 fractional format, ready to be copied and hard coded in the microprocessor's program memory. Eight different sets of parameters are used in this study. Thus, the application has eight different choices represented by the eight buttons at the end of the application's main window, as depicted in Figure 50. The program operation is simple; The user chooses a parameter to be converted by activating one of the eight buttons, waits for the program to show the parameters in their original arithmetic format in the left box, activates the button "Translate," waits for the program to do the conversion, depict the parameters in hexadecimal format in the right box, and copy and paste the converted parameters to the microprocessor's program memory.

Note the DCT coefficients, the logarithms, the mel filter-bank filter's coefficients, and the hamming window weights are calculated internally by the application. In contrast, the code book quantization vectors, *A, B* and π HMM parameters are fetched by text files stored by the MATLAB application used for training the DHM models used in this study. These text files are stored in the "C:/Thesis" directory as:

- "*codebook12x128.txt*" for the codebook quantization vectors,
- "*HMM_Am.txt*" for the *A* parameters of the seven DHMM,
- "*HMM_Bm.txt*" for the *B* parameters of the seven DHMM,
- "*HMM_Pm.txt*" for the π parameters of the seven DHMM

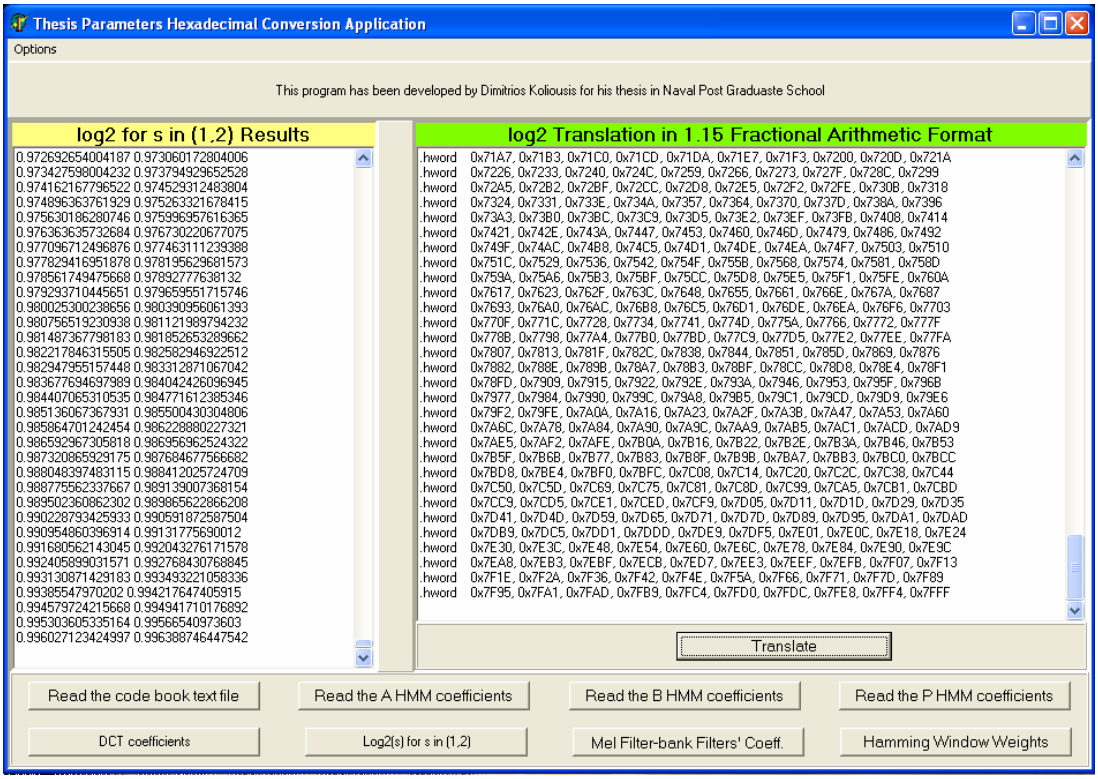

Figure 50. The "*Param_Conv_1_15_Fract_App.exe*" Application's Main Window

APPENDIX D. DEFINITION OF SEMAPHORES, FLAGS, AND GLOBAL VARIABLES USED IN THE REAL-TIME IWR SYSTEM PROGRAM IMPLEMENTATION

1. FLAGS

Name	Type	Initial Value	Description
Frame40CompleteFlag	Flag	Cleared	Section IV.B.2-3
FoundTmpStPointFlag	Flag	Cleared	Section V.A.3
FinalStartpointFlag	Flag	Cleared	Section IV.B.3 Section V.A.3
FoundTmpEndpointFlag	Flag	Cleared	Section V.A.3
FoundFinalEndPointFlag	Flag	Cleared	Section V.A.3
InitialThresholdFlag	Flag	Cleared	Section IV.B.3 Section V.A.2-3
FirstTimeFEPFlag	Flag	Set	Section VI.B.2
ReceivedDataFlag	Flag	Cleared	It is used internally by the UART2 Receiver interrupt service routine to detect the acknowledgement sent by the computer for the completion of PC and microprocessor communication
DCIDataAvailableFlag	Flag	Cleared	Section IV.B.3 Section VIII.D
EndPointDetHasRunFlag	Flag	Cleared	Section IV.B.3 Section V.A.3 Section VI.B.1-2
EndofBufferFlag	Flag	Cleared	Section IV.B.3
DCISemaphore	Semaphore	Cleared	It is used by the DCI interrupt service routine and the "KeepLast40Samples" and "DeleteFirst_nx40_Samples" routines to avoid data loss. The latter two routines delete data from the word's data buffer while the DCI interrupt saves data in this buffer. Thus it is critical for the two routines to run after the DCI interrupt is completed.

Table D.1 Flag and Semaphore Definitions

2. GLOBAL VARIABLES

Name	Values Range	Initial Value	Definition
FEPFrameStartPoint	The word's buffer available addresses	0	Section VI.B.1-2
FrameStartPoint	The word's buffer available addresses	0	Indicates the current frame's start point position in the word's buffer
Curr80FrameIndex	1 to 48	0	Indicates the current 80-sample 50% overlap speech captured frame
FrZCRCurValue	Any	0	Value of current frame's ZCR
ZCR_mean	Any	0	Mean value of frames ZCR (Section V.A.2)
ZCR_std	Any	0	Standard Deviation of frames ZCR (Section V.A.2)
ZCR_T	Any	0	ZCR Threshold (Section V.A)
FrSTECurValue	Any	0	Value of current frame's STE
STE_mean	Any	0	Mean value of frames STE (Section V.A.2)
STE_std	Any	0	Standard Deviation of frames STE (Section V.A.2)
STE_UT	Any	0	Upper-STE threshold (Section V.A)
STE_LT	Any	0	Lower-STE threshold (Section V.A)
STE_IT	Any	0	Intermediate-STE threshold (Section V.A)
ThresholdFrameIndex	1-99	0	Indicates the total number of the 80 samples 50% overlap frames, captured for the threshold calculations
DCISamplesCount	The word's buffer available addresses	0	Indicates the current total number of the captured speech samples by the DCI interrupt service routine
BufferEnd	The word's buffer last element address	0	Indicates the word's buffer last element address. Used for detection of the end of the word's buffer
SamplesCounter	1-40	0	Indicates the number of captured samples after the last 40 samples have been captured. It is cleared every time 40 samples have been captured.

Table D.2 Global Variable Definitions

3. BUFFERS AND POINTERS

Name	Buffer's Pointers	Length (bytes)	Definition
SpInputSignalBuf		12,000	Used to store the spoken word's captured samples
	SpInputSignalBufPtr		Used to indicate the current word's data buffer end address
	Frame SpInputSignalBufPtr		Used to indicate the current word's data buffer end address before the FEAP was activated.
FrSTEBuf	Buffer	50	Used to store the calculated values of frame's STE
	FrSTEBufPtr		Used to indicate the current frame's STE buffer end address
FrZCRBuf	Buffer	50	Used to store the calculated values of frame's ZCR
	FrZCRBufPtr		Used to indicate the current frame's ZCR buffer end address
FilteredOutput	Buffer	1024	Used as temporary buffer to store the calculated FFT complex numbers of the current 256-frame
ANxInputSignal	Buffer	1024	Used as temporary buffer to store the complex numbers of the current 256-frame before the FFT
Magnitude	Buffer	512	Used as temporary buffer to store the magnitude of the frequency bins calculated by the FFT for the current 256-frame
TempMelBuf	Buffer	48	Used as temporary buffer to store the results of passing the 256 frequency bins thru the mel filter-bank filters for each 256-frame
TmpDCTBuff	Buffer	28	Used as temporary buffer to store the results of the DCT during the feature extraction procedure for the current 256-frame
MFCCFeatBuf	Buffer	1344	Used to store the MFCC parameters for all the 256-frames of the spoken word during the sampling operation mode of the program.
	MFCCFeatBufPtr		Used to indicate the current MFCCFeatBuf buffer end address
ATempBuf	Buffer	128	8x8 x 2
C_buf	Buffer	16	1x8 x2

PmTempBuf	Buffer	16	1x8 x 2
aaBuf	Buffer	16	1x8 x 2
ScaleBuf	Buffer	16	1x8 x 2
LogScaleBuf	Buffer	14	1x 7 x2

Table D.3 Buffer and Pointer Definitions

4. PROGRAM MEMORY BUFFERS USED FOR PARAMETERS PROGRAMMING

Name	Program Memory Section	Length (bytes)	Stored Parameters
cb_12x128DoubleCoeff	CodeBook_coeff	3072	128 Codebook quantization vectors (128x12)
dct_cos14_26coeff	dct_cos_coeff	672	DCT Coefficients (24x14)
hamming_256coeff	hamming_coeff	512	Hamming window weights (256)
HMM_Am8x8_x7mod_coef	HMM_Am_coeff	896	A parameters of seven DHMM ((8x8) x 7)
HMM_Bm128x8_x7mod_coef	HMM_Bm_coeff	14336	B parameters of seven DHMM ((128x8) x 7)
HMM_Pm8x1_x7mod_coef	HMM_Pm_coeff	112	π parameters of seven DHMM ((8x1) x 7)
global log2_coeff_1_1999	log2_coeff_section	3998	Logarithm lookup table of 1999 numbers in (1,2)
global mel_bank24_indexes	MELBank_coeff	48	Indices of the non zero coefficients of the mel filter-bank filters
global mel_bank24_coeff		486	The actual mel filter-bank filter coefficients (243)

Table D.4 Program Memory Buffer Definitions

APPENDIX E. REAL-TIME IWR SYSTEM ASSEMBLY ROUTINES

The real-time IWR system software was implemented in a modular form. All the tasks have been programmed in separate routines for easy debugging and future updates. All routines that compose the system are listed in Table E.1, which includes the routine's calling name, assembly file name, time (in pics) needed to be executed, and a brief description of the task.

Calling Name	Assembly File Name (.s)	pics	Definition
_clear_initial_flags	Clear_Initial_Flags	34	Set the flags, semaphores, and global values to their initial value.
_clock_change	Clock8000	20	Initializes the microprocessor's clock rate
_clock_change_UART	Clock8000	20	Initializes the microprocessor's clock rate properly for UART transmission
_code_book_proc	CodeBook_Proc.s	4,000	Finds the code in the codebook that will represent the current feature vector using the Euclidean distance
_Curr_80_frame_Index	Curr80frameIndex_ Calculation	35	Calculates the current total number of the captured 80-sample 50% overlap frames
_dct_proc	dct_proc	1,100	Calculates the DCT of the 24 mel frequency bins
_delete_first_nx40_ samples	DeleteFirst_nx40_ Samples	max 1,000	Realigns the buffers and indices when a start-point has not been detected for seven 40-sample frames
_end_point_refinement	Endpoint_ refinement_proc	880	Seeks the end-point based on the ZCR after a temporary end-point has been detected
EP_Frame_STE_Calc	EP_Frame_STE_ Calculations	900	Calculates the STE of the current 80-sample 50% overlap frame. It is deactivated after a final start-point has been detected
EP_Frame_ZCR_Calc	EP_Frame_ZCR_ Calculations	900	Calculates the ZCR of the current 80-sample 50% overlap frame. It is deactivated after a final start-point has been detected

_Feature_extraction_ activation	Feature_Extr_ Activation	980	Controls the activation of the feature extraction routine
_Feature_extraction_ proc	FeatureExtraction_ proc	1,070	Feature extraction routine. It is controlled by the FEAP
_Frame_STE_Calc	Frame_STE_ Calculations	660	Calculates the STE of the current 80-sample 50% overlap frame. It is activated only during the threshold calculation procedure
_Frame_ZCR_Calc	Frame_ZCR_ Calculations	900	Calculates the ZCR of the current 80-sample 50% overlap frame. It is activated only during the threshold calculation procedure
_Frame_Threshold_ Calcs	Frame_Threshold_ Calculations	510	Calculates the thresholds
_hamming_proc	Hamming_proc	792	Windows the 256-sample frame
_init_codec_slave	Init_Codec_Slave	min. 515	Initializes the Audio Codec – DCI communication protocol
_init_dci_master	Init_DCI_Master	30	Initializes the Audio Codec – DCI communication protocol
_init_port	Init_Port	20	Initializes the microprocessor's ports as input or output
_init_uart2	Init_Uart2	10	Initializes the microprocessor's UART 2 I/O
__DCIInterrupt	Isr_DCI	50	DCI interrupt service routine
__U2RXInterrupt	Isr_UART2_RX	20	UART2 Receiver interrupt service routine
_keep_last40_ samples	_KeepLast40 Samples	80	Realign the buffers and indices when the start-point detection routine has to delete the first 40 captured samples
_log2magn	Log2Magnitude	10	Calculates the logarithm of the 24 mel frequency bins for each 256-sample framef
_log2_proc	Logarithm_proc	3,340	Calculates the logarithm of an integer
_log_likelihood_ proc	LogLikelihood_ proc	1200	Calculates the log-likelihood of the current 256-sample frame's features vector code for each of the 7 DHMM models
_magnitude	Magnitude_Proc	780	Calculates the magnitude of the FFT bins of the current 256-sample frame
_main	Main	200	Main Program

__reset	Main	5	Initializes the Stack Pointer
__OscillatorFail	Main	4	Oscillator Fail trap routine
__AddressError	Main	4	Address Error trap routine
__StackError	Main	4	Stack Error trap routine
__MathError	Main	4	Declare Math Error trap routine
_matrix_mult_8x8_ 8x1_proc	Matrix_Multiplic_ Proc	330	Matrix multiplication Routine. The execution time refers to (8x8)*(8x1) matrix multiplication
_melbank_proc	Melbank_proc	7,300	Passes the 128 frequency magnitudes of each 256-sample frame thru the mel filter-bank filters
_normal_proc	Normal_procedure	65	Controls and activates all the routines of the real-time IWR program. Runs after the initialization procedure and after the thresholds have been calculated
_refStartPoint_Proc	RefStartPoint_ Procedure	max 880	Seeks for the start-point based on the ZCR after a temporary start-point has been detected
_Int_SQRT_Calc	SQRT_Algorithm	68	Calculates the Square root of an integer
_sub_fft	Sub_FFT	7,000	Calculates the 256 FFT bins of the current 256-sample frame
_threshold_Proc	Threshold_ Procedure	44	Controls and activates the routines for the thresholds calculation. Runs after the initialization procedure and only until the thresholds calculation have been completed.

Table E.1 Real-time IWR System Assembly Global Routines

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX F. FRACTIONAL 1.15 ARITHMETIC FORMAT

The 1.15 fractional arithmetic format was used in this study to represent numbers in binary format. Fractional data is represented as a two's complement fraction where the most significant bit (MSbit) is defined as a sign bit and the radix point is positioned just after the sign bit (Q.X format). The range of a 16-bit two's complement fraction with this implied radix point is from -1.0 to $(1-2^{1-16})$ or -1.0 to 0.999969482 [Microchip, 2006]. Figure 51 shows examples of numbers expressed in 1.15 format fractional format, as interpreted by the microprocessor's arithmetic logic using the 1.15 factional format.

In addition, 16-bit signed two's complement binary arithmetic was used to represent integer numbers. Integer numbers that can be represented in this fashion range between -2^{N-1} to $2^{N-1}-1$. Figure 52 shows examples of integer representation in 16-bit signed two's complement binary numbers.

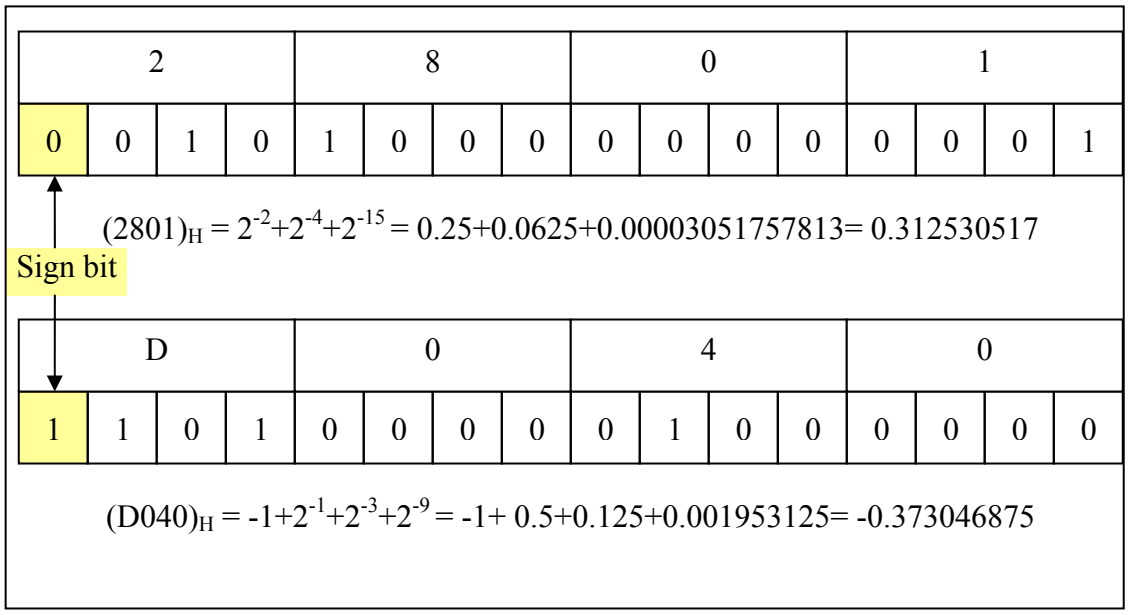

Figure 51. Fractional 1.15 Format

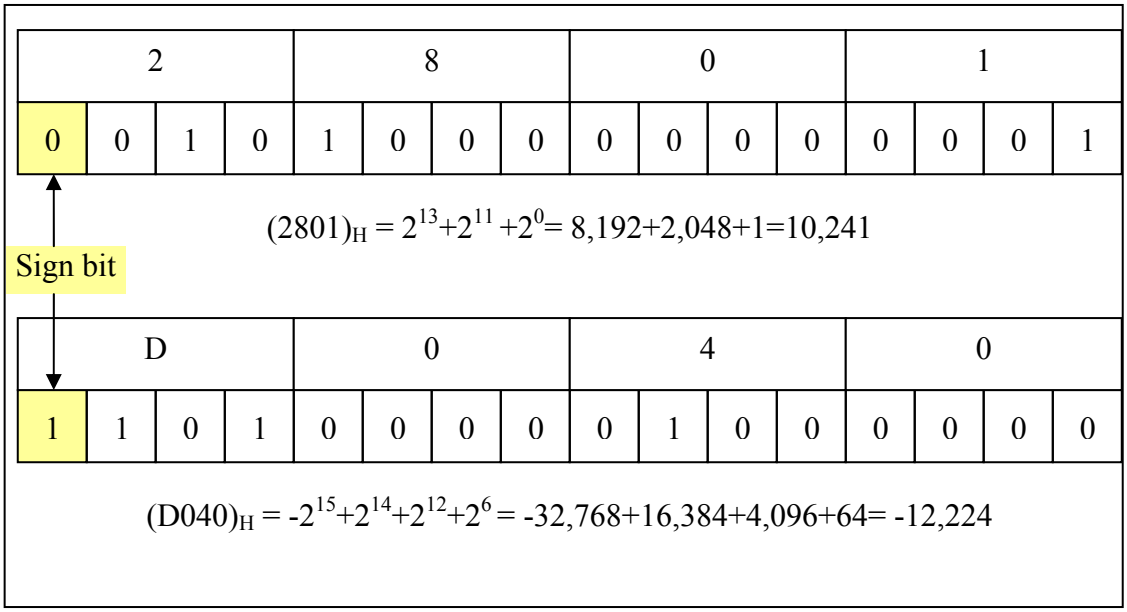

Figure 52. 16-bit Signed Two's Complement Integer Format

LIST OF REFERENCES

[Bahl et al, 1984] L. R. Bahl, S. K. Das, P. V. DeSousa et al., "Some experiments with large vocabulary isolated word sentence recognition," Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, Vol. 2, paper 26.2, 1984.

[Baum, 1966] L. E. Baum, and T. Petrie, "Statistical inference for probabilistic functions of finite state Markov chains," Annals of Mathematical Statistics, Vol. 37, pp. 1554-1563, 1966.

[Baum, 1970] L. E. Baum, T. Petrie, and G. Soules, "A maximization technique in the statistical analysis of probabilistic functions of Markov chains," Annals of Mathematical Statistics, Vol. 41, pp. 164-171, 1970.

[Becchetti, 1999] C. Becchetti, and L. P. Ricotti, Speech Recognition Theory and C++ Implementation, John Wiley & Sons, West Sussex, England, 1999.

[Bulbuller, 2006] G. Bulbuller, "Recognition of In-ear Microphone Speech Data Using Multi-layer Neural Networks," MSEE Thesis, Naval Postgraduate School, Monterey, California, March 2006.

[Cristi, 2004] R. Cristi, Modern Digital Signal Processing, Thomson/ Brooks/ Cole, Pacific Grove, California, 2004.

[Davis, 1980] S. Davis, and P. Mermelstein, "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences," IEEE Transactions on Acoustics, Speech and Signal Processing, Vol. 28, No. 4, pp. 357-366, 1980.

[Deller, 2000] J. R. Deller, J. H. L. Hansen, and J. G. Proakis, Discrete-Time Processing of Speech Signals, IEEE Press, New York, 2000.

[Deng, 2003] L. Deng, and D. O'Shaughnessy, Speech Processing: A Dynamic and Optimization-Oriented Approach, Marcel Dekker, New York, 2003.

[Dugad, 1996] R. Dugad, and U.B. Desai, "A tutorial on hidden Markov models," Technical Report No. SPANN-96.1, Indian Institute of Technology, Bombay, India, May 1996.

[Fargues, 2005] M. P. Fargues, Class Notes for EC4440 (Statistical Digital Signal Processing), Naval Postgraduate School, Monterey, California, 2005 (unpublished).

[Gold, 2000] B. Gold, and N. Morgan, Speech and Audio Signal Processing, John Wiley & Sons, 2000.

[Graciarena, 2003] M. Graciarena, H. Franco, K. Sonmez, and H. Bratt, "Combining standard and throat microphones for robust speech recognition," IEEE Signal Processing Letters, Vol. 10, No. 3, pp. 72-74, March 2003.

[Hanauer, 1971] L. S. Hanauer, and B. S. Atal, "Speech analysis and synthesis by linear prediction of the speech wave," Journal of the Acoustic Society of America, Vol. 50, pp. 637-655, 1971.

[IXCESSORY, 2007] IXCESSORY, XEM98D Earbone Microphone, [http:// www.ixcessory.com], last accessed 27 May 2007.

[Jayant, 1984] N. S. Jayant, and P. Noll, Digital coding of waveforms: principles and applications to speech and video, Prentice-Hall, 1984.

[Kang and Coulter, 1976] G. S. Kang, and D. C. Coulter, "600 bits per second voice digitizer (linear predictive formant vocoder)," Naval Research ,Laboratory Report, 1976.

[Kurcan, 2006] R. S. Kurcan, "Isolated Word Recognition from In-ear Microphone Data Using Hidden Markov Models," MSEE Thesis, Naval Postgraduate School, Monterey, California, March 2006.

[Markel, 1972] J. D. Markel, "Digital inverse filtering: A new tool for formant trajectory estimation", IEEE Transactions on Audio and Electroacoustics, Vol. 20, pp. 129-137, June 1972.

[Markel and Gray, 1976] J. D. Markel, and A. H. Gray, Linear Prediction of Speech, Springel-Verlag, New York, 1976.

[Microchip, 2006] Microchip Technology Inc., dsPIC33F Family Data Sheet; High-performance 16-bitDigital Signal Controllers, DS70165D Edition, July 2006.

[Milner, 1970] P. M. Milner, Physiological Psychology, Holt, Rinehart and Winston, New York, 1970.

[Nacre, 2007] Nacre AS, QUIETPRO Headphones, [http://www.nacre.no], last accessed 05.27.2007.

[Noll, 1967] A. M. Noll, "Cepstrum pitch determination," Journal of the Acoustical Society of the America, Vol. 41, pp. 293-309, February 1967.

[O'Shaughnessy, 1987] D. O'Shaughnessy, Speech Communications, Addison Wesley, 1990.

[Parhami, 2000] B. Parhami Computer Arithmetic: Algorithms and Hardware Designs, Oxford University Press, New York, 2000.

[Picone, 1993] J. W. Picone, "Signal modeling techniques in speech recognition," Proceedings of the IEEE, Vol. 81, No. 9, pp. 1215-1247, September 1993.

[Rabiner, 1975] L. R. Rabiner, and M. R. Sambur, "An algorithm for determining the endpoints of isolated utterances," The Bell System Technical Journal, February 1975.

[Rabiner, 1989] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," Proceedings of the IEEE, Vol.77, No.2, pp.257- 286, February 1989.[Rabiner, 1993] L. R. Rabiner and B-H. Juang, Fundamentals of Speech Recognition, Prentice Hall, 1993.

[Schafer and Rabiner, 1978] L. R. Rabiner, and R. W. Schafer, Digital Processing of Speech Signals, Prentice-Hall, Englewood Cliffs, New Jersey, 1978.

[Si3000 Data Sheet, 2000] Silicon Laboratories Inc., Si3000 Voice Band Codec with Microphone/Speakers Drive, Si3000-DS11 Edition, Rev.1.1, 2000.

[Theodoridis, 2003] S. Theodoridis and K. Koutroumbas, Pattern Recognition, Second Edition, Academic Press, San Diego, California, 2003.

[Therrien, 2004] C. W. Therrien, M. Tummala, Probability for Electrical and Computer Engineers, CRC Press, 2004.

[Tummala, 2007] M. Tummala, Class Notes for EC4430, Multimedia Information and Communications, Naval Postgraduate School, Monterey, California, 2005 (unpublished).

[Vaidyanathan, 2004a] R. Vaidyanathan, K. Hyunseok, L. Gupta and J. West, "Parametric and non-parametric signal analysis for mapping air flow in the ear-canal to tongue movements: a new strategy for hands-free human-machine interfaces," Proceedings of the 2004 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP `04), Vol. 5, pp. 613-16, May 2004.

[Vaidyanathan, 2004b] R. Vaidyanathan, L. Gupta, B. Chung, T. J .Allen, R. D. Quinn, M. Tabib-Azar, J. Zarycki, and J. Levin, "Human-machine interface for tele-robotic operation: mapping of tongue movements based on aural flow monitoring," Proceedings of 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2004), Vol. 1, pp. 859-865, September - October 2004.

[Westerlund, 2003] N. Westerlund, "Applied Speech Enhancement for Personal Communication," Licentiate thesis, Blekinge Institute of Technology, Ronneby, Sweden, May 2003.

[Zhang, 2004] Z. Zhang, L. Zicheng, M. Sinclair, A. Acero, L. Deng, J. Droppo, X. Huang and Y. Zheng, "Multi-sensory microphones for robust speech detection, enhancement and recognition," Proceedings of the 2004 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP `04), Vol. 3, pp. iii-781-4, May 2004.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. General Staff/Dep. B/3
   Hellenic Air Force
   Athens, Greece

4. General Staff/Dep. C/3
   Hellenic Air Force
   Athens, Greece

5. Hellenic Air Force Academy
   Hellenic Air Force
   Tatoi, Greece